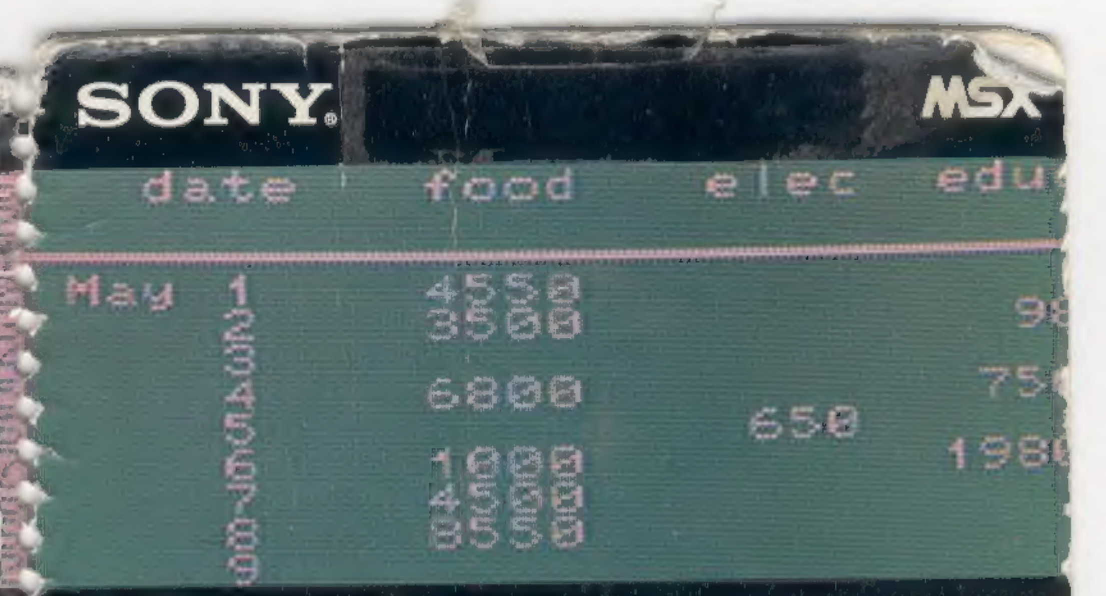


Aprenda a programar en BASIC-MSX

HIT BIT

CON PROGRAMAS



APRENDA A PROGRAMAR EN BASIC MSX

KANDAHAR

SOFT



3ª Edición

Sabino de Arana 42-44
08028-Barcelona

Dpto. Legal B: 3940-87
ISBN 84-398-4421-2
Imprime: Gráficas Signo, S.A.
Carretera de Cornellá, 140, 2º – Esplugues de Llobregat
Printed in Spain. Impreso en España.

COD. 21000195

PROLOGO

Hasta la aparición del MSX, los ordenadores existentes en el mercado se caracterizaban por su incompatibilidad. Los programas y los distintos periféricos conectables a la unidad central eran solamente válidos para aquella marca que estaban previstos.

Esta incompatibilidad era, y es, un grave inconveniente para el usuario final, que queda indefenso ante los vaivenes que pueda sufrir cada una de estas empresas.

El MSX ha revolucionado el mercado en este aspecto. Son ya más de veinte compañías que han adoptado este standard, dispuestas a terminar con la anarquía de lenguajes y de conexión de periféricos existentes.

Se pretende así ofrecer una amplia gama de programas, ordenadores y periféricos para que el usuario pueda escoger aquello que mejor se adapte a sus necesidades de entre todos los modelos existentes, contando con la sguridad de que podrá adquirir programas y periféricos de cualquiera de las compañías MSX.

Pero esta es sólamente una de las ventajas del MSX.

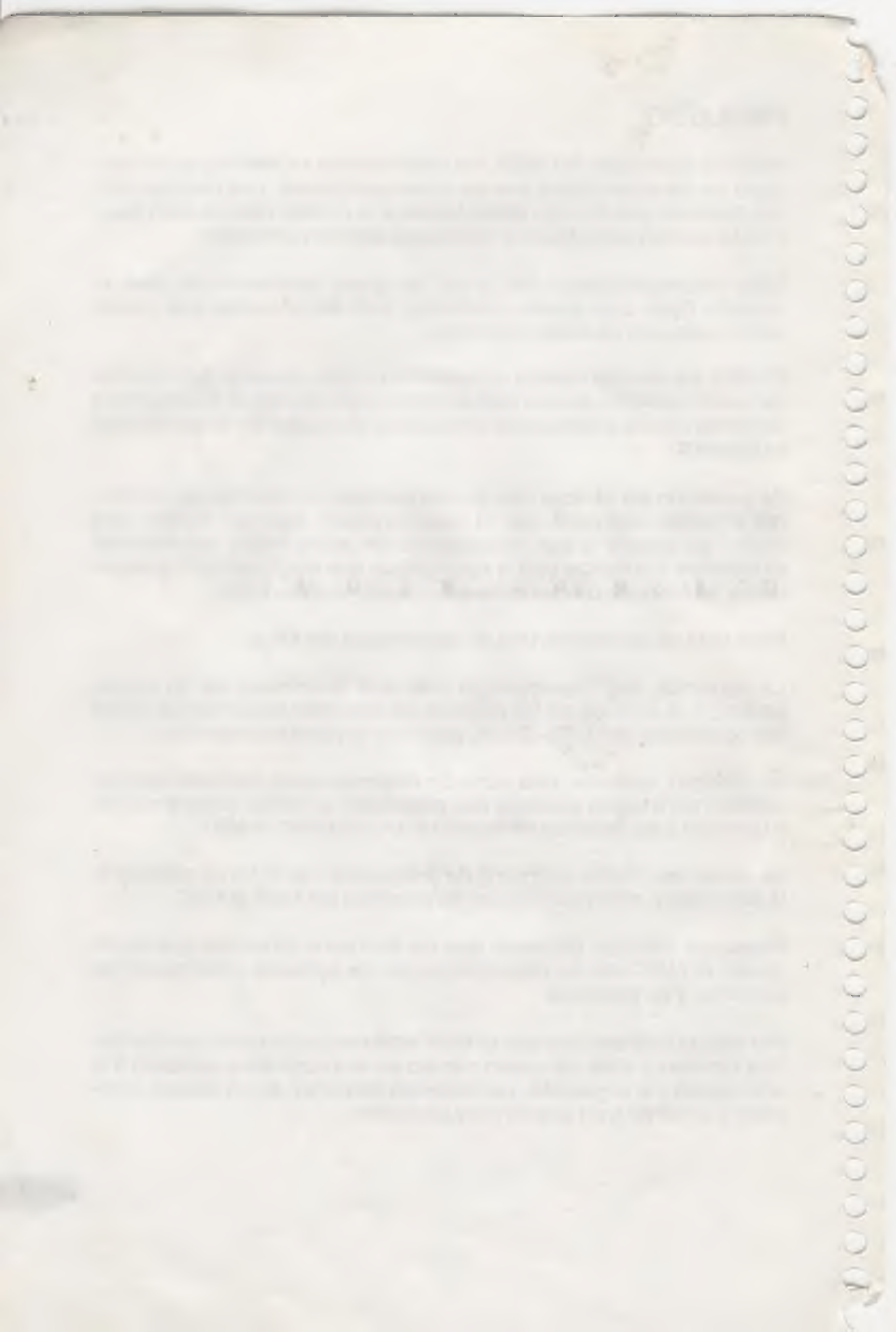
La segunda, tan importante o más que la primera, es su propia potencia. A lo largo de las páginas de este libro se comentan todos los comandos del MSX-BASIC para que lo pueda comprobar.

Se ofrecen, además, una serie de informaciones técnicas de gran utilidad para todos aquellos que pretendan aprender programación y también para quienes ya desean profundizar en el MSX.

Se listan, asimismo, una serie de programas con el fin de demostrar la sencillez y, al propio tiempo, la potencia del MSX-BASIC.

Podemos concluir diciendo que las tres características que configuran el MSX son su estandarización de software y hardware, su sencillez y su potencia.

Por ello puede decirse que el MSX está revolucionando la informática familiar y abre un nuevo campo en la informática aplicada a la educación y a la gestión, permitiendo disponer de un equipo completo y potente a un precio muy asequible.



INDICE:

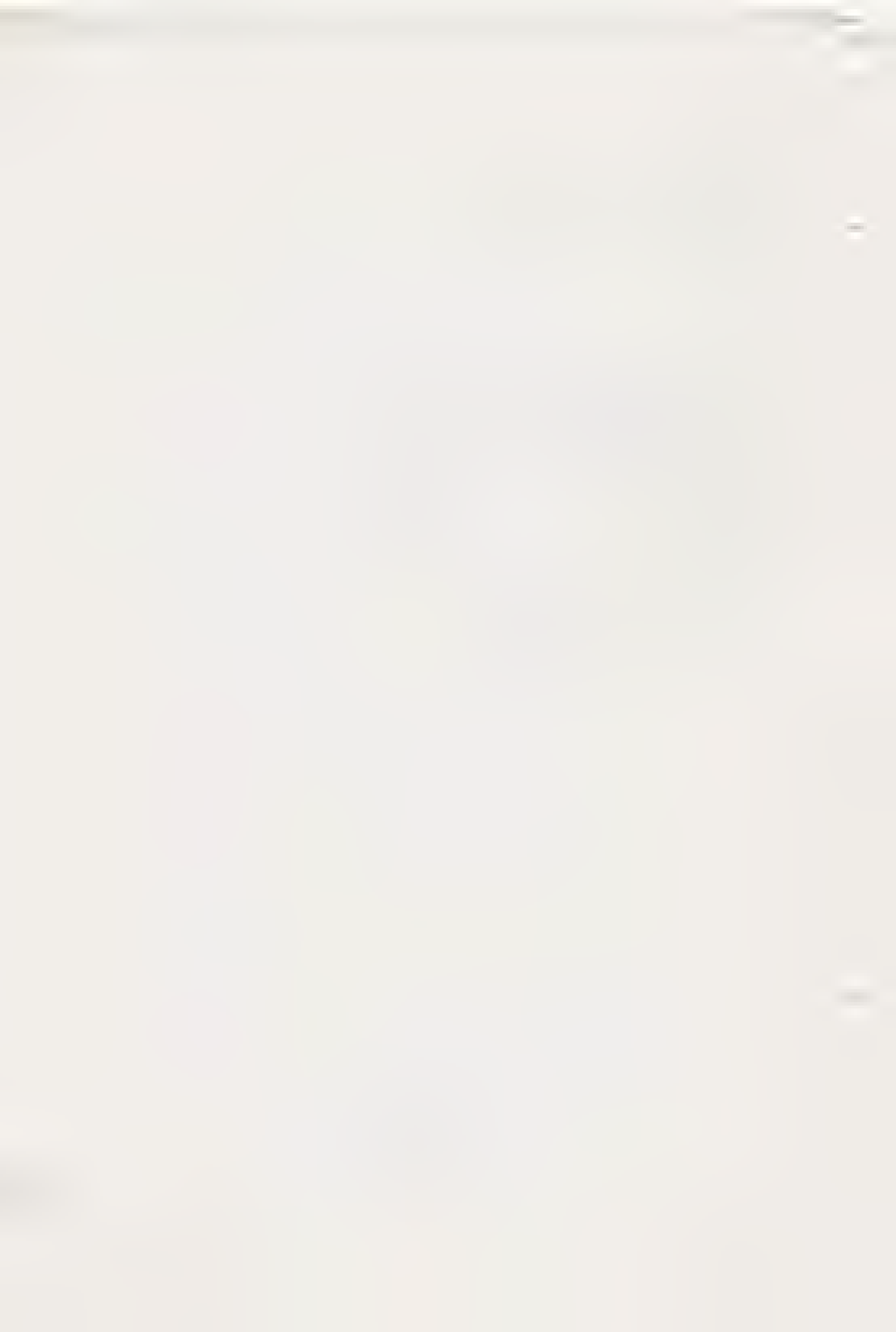
PRIMERA PARTE

INFORMACION GENERAL DEL BASIC MSX

SEGUNDA PARTE

COMANDOS, SENTENCIAS Y FUNCIONES DEL BASIC MSX

ANEXOS:



PRIMERA PARTE

INFORMACION GENERAL DEL BASIC MSX



1.0. INTRODUCCION

El **BASIC MSX** es una versión ampliada del Microsoft Standard Basic version **4.5**, e incluye soportes para gráficos, musica y diversos periféricos

Fue disenado para seguir el **GW-BASIC**, estandard en el mundo de los microprocesadores de 16 bits Pero el mayor esfuerzo fue hacer todo el sistema lo mas flexible y ampliable posible

El **BASIC MSX** trabaja con 14 digitos para ofrecer doble precision en las funciones matematicas por lo que no se generaran más esos extranos errores de redondeo que confunden a los usuarios

1.1. MODOS DE FUNCIONAMIENTO

Una vez inicializado el **BASIC MSX**, aparece rapidamente el OK «OK», indica que el sistema está dispuesto para aceptar cualquier comando Es ahora cuando el **BASIC MSX** puede ser usado en modo directo o indirecto.

En el modo directo los comandos no estan precedidos por numero de linea y se ejecutan al ser introducidos Los resultados de operaciones aritmeticas y logicas aparecen inmediatamente en la pantalla y son almacenados para su uso posterior, pero las instrucciones desaparecen tras su ejecucion (no quedan almacenadas en memoria)

El modo directo es util para utilizar el **BASIC MSX** como calculadora, para rapidas operaciones que no requieren un programa completo

El modo indirecto se utiliza para introducir programas Las lineas del programa están precedidas por numeros y son almacenadas en memoria El programa almacenado en memoria será ejecutado al introducir el comando **RUN.**

1.2. FORMATO DE LAS LINEAS

En una linea de programa pueden situarse varias instrucciones, siempre que esten separadas por dos puntos

10 instruccion **BASIC** instrucción **BASIC**

Toda linea de programa comienza siempre con un numero de linea y termina al pulsar **RETURN**, pudiendo contener un maximo de 255 caracteres.

1.2.1. Numero de linea

Cada linea de un programa **BASIC MSX**, comienza con un numero.

Los numeros de linea indican el orden en que han sido almacenadas en memoria las lineas del programa

También son utilizados como referencia en la edición del programa y en las sentencias de bifurcación.

Los numeros de linea deben estar comprendidos entre 0 y 65 529

1.3. CONJUNTO DE CARACTERES

El conjunto de caracteres del **BASIC MSX** esta formado por caracteres alfabeticos, caracteres numericos, caracteres especiales y carácteres gráficos.

Los caracteres alfabeticos son las letras del alfabeto

Los caracteres numericos son los digitos del 0 al 9

Los siguientes caracteres especiales estan reconocidos por el **BASIC MSX**:

Caracter	Acción
=	signo de equivalencia
+	suma
—	resta
*	multiplicación

Caracter	Accion
/	division
^	simbolo exponencial o acento circunflejo
(	paréntesis izquierdo
)	paréntesis derecho
%	porcentaje
$	signo de dólar
!	exclamación
[	corchete izquierdo
]	corchete derecho
	punto o punto decimal
	apóstrofe
;	punto y coma
:	dos puntos
&	and
?	signo de interrogación
<	menor que
>	mayor que
Y	signo del yen

1.4. CONSTANTES

Las constantes son los valores que el **BASIC MSX** usa durante la ejecucion de un programa Hay dos tipos de constantes numéricas y alfanuméricas (string).

Una constante alfanumérica es una sucesion de hasta 255 caracteres alfanuméricos insertados entre comillas

' HOLA ' 25 000 000 $' Numero de empleados'

Las constantes numericas son numeros positivos o negativos, y no pueden contener comas (El punto realiza la función de la coma) Hay seis tipos de constantes numéricas.

1. **Constantes enteras**: todo numero entre -32768 y 32767 Las constantes enteras no contienen decimales
2. **Constantes con punto decimal fijo:** numeros reales positivos o negativos que contengan decimales.

3. **Constantes con punto decimal,** numero positivos o negativos en forma exponencial El margen para estas constantes esta entre 10^{-64} y 10^{+63}

235 988 E-7 = 0 0000235988 2359 E6 = 2359000000

4. **Constantes hexadecimales:** numeros hexadecimales con el prefijo & H

& H76 & H32 F

5. **Constantes octales:** numeros octales con el prefijo & 0 o &

& 0 347 & 347

6. **Constantes binarias:** numeros binarios, con el prefijo & B

& B 01110110 & B 11100111

1.4.1. Constantes numéricas de simple y doble precision

Las constantes numericas de simple precision estan almacenadas con 6 digitos y son impresas con hasta 6 digitos de precision Las constantes numericas de doble precision estan almacenadas con 14 digitos y son impresas con hasta 14 digitos de precisión.

Una constante de simple precisión es toda constante numerica que cumple una de las siguientes caracteristicas

1. Forma exponencial usando E
2. Signo de admiración (!)

-1 09 E-06
22 5!

Una constante de doble precisión es toda constante numerica que cumple una de las siguientes caracteristicas

1. Todo numero no exponencial (usando E) sin signos de admiración
2. Numero exponencial usando la letra D
3. Numero con el signo #

3489
345962811
-1 09432 D-06
3489.0 #

1.5. VARIABLES

Las variables son nombres usados para representar valores. El valor de una variable puede ser asignado explicitamente por el programador o puede ser el resultado de unos cálculos en el programa. Al inicializar el sistema, todas las variables tendran asignado el valor cero (variables numericas) o «null» (variables alfanumericas)

1.5.1. Nombres de variables y Declaración de caracteres

Los nombres de variables en el **BASIC MSX** pueden ser de cualquier longitud, pero solo son significativos 2 caracteres. Estas variables pueden contener letras y numeros pero el primer carácter debe ser siempre una letra.

Una variable no puede ser una palabra reservada y no puede contener palabras reservadas Las palabras reservadas incluyen todos los comandos del **BASIC MSX**, nombres de funciones ordenes y nombres operativos Por ejemplo, si una variable comienza por **FN**, se supondra que es la llamada de una funcion definida por el usuario y no el nombre de una variable

Las variables pueden representar un valor numerico o una cadena de caracteres Las variables alfanumericas se representan con el simbolo $ como ultimo caracter

A$ = "VENTAS

Las variables numericas pueden ser numeros enteros de simple o doble precision Los simbolos que identifican esas variables son

% variable entera

! variable de simple precisión.

\# variable de doble precisión

Pi #	doble precisión
Minimum!	simple precisión
Limit %	variable entera
N$	variable alfanumérica
ABC	doble precisión

Existe un segundo metodo por el cual pueden definirse los tipos de variab es utilizando las sentencias **DEFINT**, **DEFSTR**, **DEFSGN** y **DEFDBL** en el transcurso de un programa

1.5.2. **Variables matriciales** (variables subindicadas)

El **BASIC MSX**, permite trabajar con variables matriciales para designar elementos de una matriz.

Las matrices deben declararse por medio de una sentencia de

dimensionado (DIM) en el programa pero ello no es imprescindible si los subíndices no superan el rango de 0 a 10

El nombre de una variable con subíndice es siempre variable y va seguido por una o más expresiones entre paréntesis La expresión entre paréntesis indica la posición de los datos en la Matriz.

A$(4) variable subindicada de una dimensión

A(2 3) variable subindicada de dos dimensiones

A(2 3, ,6) variable subindicada de varias dimensiones

Una matriz puede tener hasta 255 dimensiones El número máximo de elementos está determinado por la capacidad de memoria

1.5.3. Requisitos de espacio

La siguiente tabla indica el número de bytes ocupados por los valores representados en los distintos tipos de variables

	Tipo	Bytes
Variables numéricas:	entera	2
	simple precisión	4
	doble precisión	8
Matrices:	entera	2 por elemento
	simple precisión	4 por elemento
	doble precisión	8 por elemento
Variables alfanuméricas:	3 bytes más el contenido de la cadena	

1.6. CONVERSION DE TIPOS

Cuando es necesario el BASIC MSX convierte una constante numérica de un tipo en otra He aquí unos ejemplos

1 Si una constante numérica de un tipo se iguala a una variable numérica distinta, el número será almacenado como se indica en la variable

```
10 A % = 23,42
20 PRINT A %
RUN
23
```

Si una variable string se iguala un valor numérico o viceversa, aparecerá el siguiente mensaje de error Type mismatch

2 Durante el cálculo de una operación todos los operandos son convertidos a un único grado de precisión y lo mismo sucede con el resultado

```
10 D = 6/7!           La operación y resultado
20 PRINT D            son en doble precisión.
RUN
. 85714285714286
```

```
10 D! = 6/7           La operación se ha hecho
20 PRINT D!           en doble precisión pero el
RUN                   resultado aparece en D!, con
. 857143              simple precisión y
                      redondeado
```

3 Los operandos pueden ser convertidos en enteros y obtener un resultado en números enteros Los operandos deben estar comprendidos entre —32768 y 32767 para que no exista un error de «overflow».

4 Cuando un valor con punto decimal flotante se convierte en un entero, la parte decimal desaparece.

```
10 C% = 55.88
20 PRINT C %
RUN
55
```

5 Si una variable de doble precisión se asigna a un valor de simple precisión sólo serán válidos los primeros 6 dígitos

```
10 A! = SQR (2)
20 B = A!
30 PRINT A!, B
RUN
1.41421       1.41421
```

1.7. EXPRESIONES Y OPERADORES

Una expresión puede ser una cadena o una constante numérica, una variable, una combinación de constantes y variables con operandos que producen un valor.

Los operadores del **BASIC MSX** pueden ser divididos en 4 clases aritméticos relacionales, lógicos y funcionales

1.7.1. Operadores aritméticos

Operador	Operación	Expresión
^	Exponencial	X ^ Y
—	Negación	— X
*, /	Multiplicación, División	X*Y X/Y
+,—	Adición, sustracción	X+Y X -Y

Para cambiar el orden de realización de las operaciones, se usa el paréntesis.

Las operaciones en el interior del paréntesis se realizan en primer lugar Dentro del paréntesis, se mantiene el orden usual de las operaciones multiplicación y división en primer lugar y tras estas, suma y resta.

1.7.1.1. División entera y Resto

Dos operandos adicionales están disponibles en el **BASIC MSX**

La division entera se indica con el simbolo de Yen Los operandos se convierten a enteros (entre —32768 y 32767) antes de efectuar la division y el cociente es redondeado a entero

```
10/4 = 2
25 68/6 99 = 4
```

MOD, da el valor entero del resto de la division

```
10 MOD 4      2 (10/4      2 con el resto 2)
25 63 MOD 6 99      4 (25/6      4 con el resto 1)
```

1.7.1.2. Overflow y división por cero

Si durante el calculo de una operación, se encuentra la division por cero aparece el mensaje de error «Division by zero» y termina la ejecución del programa

Asimismo, cuando se produce una sobrecarga de datos aparece el mensaje de error «Overflow»

1.7.2. Operadores relacionales

Son usados para comparar dos valores El resultado de la comparación es verdadero (—1) o falso (0)

Operador	Operacion	Expresión
=	Igualdad	X = Y
<>	Desigualdad	X <> Y
<	Menor que	X < Y
>	Mayor que	X > Y
<=	Menor o igual que	X <= Y
>=	Mayor o igual que	X >= Y

El signo igual se usa tambien para asignar un valor a una variable

Cuando se combinan operadores aritmeticos y relacionales en una expresion el aritmético se ejecutara siempre en primer lugar.

$$X + Y < (T-1)/2$$

En primer lugar se realizaran las operaciones X + Y y (T-1)/2, luego se compararan los resultados obtenidos

1.7.3. Operadores lógicos

Realizan operaciones logicas, bit a bit El resultado puede ser cierto (1) o falso (0) En una expresión, las operaciones logicas son ejecutadas tras las aritmeticas y las relacionales La funcion que realizan los operadores logicos NOT, AND, OR, XOR, EQU e IMP se resume en la siguiente tabla

Operadores lógicos del BASIC MSX.

NOT

X	NOT X
1	0
0	1

AND

X	Y	X AND Y
1	1	1
1	0	0
0	1	0
0	0	0

OR

X	Y	X OR Y
1	1	1
1	0	1
0	1	1
0	0	0

XOR

X	Y	X XOR Y
1	1	0
1	0	1
0	1	1
0	0	0

EQU	X	Y	X EQV Y
	1	1	1
	1	0	0
	0	1	0
	0	0	1

IMP	X	Y	X IMP Y
	1	1	1
	1	0	0
	0	1	1
	0	0	1

Así como los operadores relacionales pueden ser usados para tomar decisiones base al programa efectuado, los operadores lógicos pueden conectar dos o mas relacionales y dar un valor vedadero o falso para ser usado en una decisión

```
IF   D < 200 AND F < 40 THEN 80
IF   I > 10 OR K >0 THEN 50
IF   NOT P THEN 100
```

63 AND 16 = 16

63 = binario 111111
16 = binario 10000
así, 63 AND 16 = 16 (binario, 10000)

15 AND 14 = 14

15 = binario 1111
14 = binario 1110
asi, 15 AND 14 = 14 (binario 1110)

—1 AND 8 = B

—1 = binario 1111111111111111
8 = binario 1000
así, —1 AND 8 = 8 (binario 1000)

4 OR 2 = 6

4 = binario 100
2 = binario 10
así, 4 OR 2 = 6 (binario 110)

```
10 OR 10 = 10    10 = binario 1010
                 así 1010 OR 1010 = 1010

  1 OR 2    1    —1 = binario 1111111111111111
                 —2 = binario 111111111111111
                 así —1OR —2 = —1 (binario 1111111111111111)
NOT X = —(x+1)
```

1.7.4. Operadores funcionales

El **BASIC MSX,** tiene funciones intrinsecas residente en el sistema como **SQR** (raiz cuadrada) o **SIN** (seno) Tambien pueden definirse funciones escritas por el programador con el comando. **DEF FN.**

1.7.5. Operaciones con cadenas (strings)

Los strings pueden ser encadenados usando el signo +

```
10 A$ = "FILE": B$ = "NAME"
20 PRINT A$ + B$
30 PRINT "NEW" + A$ + B$
RUN
FILE NAME
NEW FILENAME
```

Los strings pueden ser comparados usando los mismos signos de comparacion que los usados con numeros

```
= <   <   >   <=   >=
```

Las comparaciones entre strings se efectuan carácter a caracter utilizando el código **ASCII.** Si todos los códigos **ASCII**

son iguales, los string son iguales. Si durante la comparación de strings se llega al final de uno de ellos, el string más corto será el más pequeño. Los espacios en blanco son significativos.

```
"A" < "AB"
"FILENAME" = "FILENAME"
 X&   >   X
 CL   >   CL
"kg"   < "KG"
"SMYTH" < "SMYTHE"
B$ < "9/12/83" donde B$ = "8/12/83"
```

Las comparaciones de strings pueden ser usadas para comprobar el valor de un string o para alfabetizar strings. Todas las constantes utilizadas en las expresiones de comparación deben ir entre comillas.

1.8. PROGRAMA EDITOR

El programa editor permite al usuario realizar todos los cambios necesarios dentro del programa (borrar o añadir líneas de programa, insertar instrucciones...) utilizando para ello los cursores y las teclas **INSERT** (insertar) y **DELETE** (borrado). El usuario puede situarse rápidamente en cualquier punto de la pantalla y hacer las correcciones necesarias.

Programas

El Editor de pantalla está en funcionamiento continuamente desde la aparición de «OK» hasta que se ejecuta un programa mediante **RUN.** El Editor procesa todas las líneas de texto introducidas considerando una línea de programa toda aquella que comienze por un número.

Si se intenta borrar una línea inexistente aparecerá el mensaje de error. «Undefined line number».

El Editor procesará los comandos del programa en los siguientes casos

1 **Adición de una nueva linea al programa:** El numero de lineas debe estar comprendido entre 0 y 65529 y debe ir seguido de un carácter como minimo

2 **Modificación de una linea ya existente:** la linea existente queda reemplazada por el nuevo texto introducido

3 **Borrado de una linea:** Se produce si la nueva linea contiene solo el número de línea que se desea borrar.

4. **Se produce un error.**

Si se pretende introducir una nueva linea de programa y no hay suficiente espacio en memoria, se imprimirá el mensaje de eror «Out of memory».

Una linea de programa puede contener varias instrucciones, siempre que esten separadas por dos puntos ( ) El numero maximo de caracteres por línea de programa es 250.

Edición de Programas

Mediante el comando **LIST** nos apareceran todas las instrucciones del programa, pudiendo asi ser editadas El texto puede ser modificado moviendo el cursor hasta el lugar donde se precisa hacer el cambio Esta modificacion puede realizarse efectuando una de las siguientes acciones

1 Escribiendo directamente sobre el caracter equivocado
2 Borrando los caracteres a la derecha del cursor
3 Borrando los caracteres a la izquierda del cursor
4. Insertando caracteres.
5 Anadiendo caracteres al final de la linea

Los cambios en una linea quedan grabados cuando se pulsa la techa **RETURN.**

Funciones del editor de pantalla

La siguiente tabla muestra los codigos hexadecimales del **BASIC MSX**

Tabla 1 Funciones del control del **BASIC MSX**. Pulsando a un tiempo la techa **CTRL** y otra determinada, se ejecuta una operacion especial

Codigo hexadecimal	CTRL +	Operación realizada
01	A	—
02	B	Mueve el cursor al principio de la palabra precedente
03	C	Abandona la espera de un input y la numeración automatica de lineas
04	D	—
05	E	Borra texto entre el cursor y el final de la linea.
06	F	Mueve el cursor al principio de la palabra siguiente
07	G	Sonido Bip
08	H	Misma función que tecla BS
09	I	Misma función que tecla TAB
0A	J	Mueve el cursor a la linea posterior.
0B	K	Misma función que tecla HOME
0C	L	Limpia pantalla
0D	M	Misma función que tecla RETURN.
0E	N	Mueve el cursor a la posición posterior a la del ultimo caracter de la linea
0F	O	—
10	P	—
11	Q	—
12	R	Misma función que tecla INS
13	S	—
14	T	—
15	U	Borra los caracteres de la linea hasta la posición del cursor
16	V	—
17	W	—
18	X	Misma función que tecla SELECT
19	Y	—
1A	Z	—
1B	[	Misma función que tecla ESC.
1C	/	Mueve el cursor a la derecha
1D	]	Mueve el cursor a la izquierda
1E	^	Mueve el cursor arriba
1F	—	Mueve el cursor abajo.

Funciones de teclas especiales

Tecla TAB Mueve el cursor 8 posiciones a la derecha Todos los caracteres existentes en esas posiciones son borrados

Tecla RETURN Debe pulsarse siempre que se haya terminado de entrar una linea

Tecla ESC La función de esta tecla viene determinada por el software utilizado Para el **BASIC MSX,** no tiene utilidad concreta

Tecla STOP Al presionar la tecla se interrumpe la ejecucion de un programa o de un listado Al presionarla de nuevo, se reanuda la ejecucion Pulsando al mismo tiempo CTRL y STOP se termina la ejecucion del programa con el siguiente mensaje Break in (n° instruccion)

Tecla SELECT La funcion de esta tecla viene determinada por el software utilizado Para el **BASIC MSX,** no tiene utilidad concreta

Tecla INSERT Al pulsarla, se reduce el tamano del cursor y pueden introducirse los caracteres deseados en la posición del cursor mientras los caracteres a la derecha del mismo van desplazandose a medida que se introducen los nuevos Pulsando la tecla INSERT de nuevo, el cursor vuelve a su posición normal y termina la funcion de inserción Pulsando las teclas de movimiento del cursor o pulsando RETURN, tambien termina la función de inserción.

Tecla HOME Mueve el cursor al extremo superior izquierdo de la pantalla.

Tecla DEL Borra el caracter situado en la posicion del cursor Los caracteres siguientes se desplazan una posicion a la izquierda.

Tecla BS Al pulsarla el cursor se desplaza a su izquierda, borrando el carácter que esté en esa posición.

Tecla SHIFT Pulsandola al mismo tiempo que cualquier tecla, aparece en pantalla el simbolo superior izquierdo de la tecla en cuestión

Tecla CAP Las letras que aparecen en pantalla son mayusculas, pero los numeros y símbolos no varian.

Tecla CODE Pulsando CODE y cualquier tecla aparece el simbolo inferior izquierdo de la tecla Si pulsamos, además, SHIFT, aparece el simbolo superior izquierdo

Tecla GRAPH Pulsándola al mismo tiempo que cualquier tecla, aparece el símbolo inferior derecho de la tecla. Si pulsamos además, SHIFT aparece el símbolo superior derecho.

1.9. TECLAS DE FUNCION

El **BASIC MSX**, dispone de 10 funciones predefinidas en las teclas de función F1/F10. El contenido de estas teclas se visualiza en el margen inferior de la pantalla pudiendo ser reprogramadas mediante el comando **KEY.**

Las funciones predefinidas son:

F1	color
F2	auto
F3	goto
F4	list
F5	run

Al pulsar la tecla **SHIFT**, tendremos

F6	color 15,4,7 RETURN
F7	cload "
F8	cont RETURN
F9	list RETURN
F10	cls run RETURN

SEGUNDA PARTE

COMANDOS, SENTENCIAS Y FUNCIONES DEL BASIC MSX



1. COMANDOS DE AYUDA AL PROGRAMADOR

- **AUTO**
- **DELETE**
- **LIST**
- **LLIST**
- **NEW**
- **RENUM**
- **KEY LIST**

AUTO <num. línea>, <incremento>

Numera automaticamente las lineas de un programa

- <num. linea> linea de inicio del programa
- <incremento>: salto entre lineas

AUTO empieza la numeracion en <num. linea> y va numerando las siguientes segun el valor de <incremento>

Si **AUTO** genera un numero de linea utilizado anteriormente con otra instruccion aparecera un asterisco (*) Pulsando **RETURN**, conservara el contenido de la linea y generara una nueva De lo contrario cualquier entrada que se efectue en esa linea, anulara la anterior

Para anular el comando **AUTO**, pulsar **CTRL - STOP** o **CTRL-C.**

AUTO Numera desde la línea 10 con incrementos de 10

10
20
30
:

AUTO 120 Numera desde la línea 120 con incrementos de 10

120
130
140
:

AUTO 100,5 Numera desde la línea 100 con incrementos de 5

100
105
110
:

AUTO,5 Numera desde la línea 0, con incrementos de 5

0
5
10
:

DELETE < num. línea a> — <num. linea b>
Borra las líneas de programa comprendidas entre < num línea a> y <num línea b>.

- <num. línea a>: línea de inicio de borrado
- <núm. línea b>: última línea a borrar.

Si el número de línea especificado en <num línea b> no existe en el programa, se producirá el error «Illegal function call»

DELETE 10-60 Borra desde la línea 10 hasta la 60 (ambas inclusive)

DELETE 20. Borra la línea 20.

NOTA - También puede borrar una sola línea escribiendo el num de línea y pulsando **RETURN**:

20 (RETURN).

LIST <nº línea a> - <nº línea b>
Lista el programa por pantalla.

- <nº línea a>: línea de inicio del listado
- <nº línea b>. línea final del listado.

Podemos detener momentáneamente el listado, pulsando **STOP.** Pulsando **STOP** de nuevo continuará el listado

Para anular la función **LIST,** pulsar **CTRL-STOP.**

LIST	: lista todo el programa.
LIST 10	: lista la línea 10 del programa,
LIST 10-50	: lista desde la línea 10 hasta la 50 (ambas inclusive)
LIST -50	: lista hasta la línea 50
LIST 50-	lista desde la línea 50 hasta el final del programa

LLIST <nº línea a> - <nº línea b>
Lista el programa por impresora. Funciona igual que **LIST.** (ver **LIST**)

NEW Borra el contenido de la memoria **RAM** de usuario

RENUM <nº línea nuevo> <nº línea actual>, <incremento>
Cambia la numeración de las líneas de un programa

- <nº línea nuevo> línea inicial de la nueva numeración
- <nº línea actual> línea inicial de la numeración actual
- <incremento> Salto entre líneas que se utilizará en la nueva numeración. Si no se especifica, será 10

RENUM cambia también las referencias de los números de línea que acompañan a **GOTO, GOSUB, THEN, ELSE, ON GOTO, ON GOSUB** y **ERL.**

RENUM	Renumera el programa desde la línea 10 con incrementos de 10.
RENUM 1000	Renumera y traslada el programa a partir de la línea 1000 con incrementos de 10.
RENUM 100, 60 5	Traslada a la línea 100 numerando con incrementos de 5, el programa que estaba en la línea 60

```
10 REM $ EJEMPLO RENUM $
20 PRINT "HIT BIT"
30 PRINT "BASIC"
50 PRINT "MSX"
60 A = 8
70 PRINT A
77 B = 10
85 PRINT B
90 GOTO 70
```

```
RENUM 1000, 60, 5

10 REM $ EJEMPLO RENUM $
20 PRINT "HIT BIT"
30 PRINT "BASIC"
50 PRINT "MSX"
1000 A = 8
1005 PRINT A
1010 B = 10
1015 PRINT B
1020 GOTO 1005
```

REM <comentarios>
Es un comando no ejecutable por el ordenador que permite hacer comentarios dentro de un programa.

●<comentarios> Comentarios introducidos para diferenciar las distintas partes de un programa, siendo util unicamente en los listados ya que **REM** no es ejecutado por el ordenador

```
10 REM PROGRAMA PRINCIPAL
100 REM SUBRUTINA 1
600 REM CONTADOR
```

KEY LIST

Lista el contenido de todas las teclas de funcion (F1-F10)

```
KEY LIST (RETURN)
      color
      auto
      goto
      list
```

```
run
color 15 4,7
cload
cont
list
run
```

Color corresponde a la tecla F1 Auto a F2, Goto a F3 y así sucesivamente, hasta F10

2. COMANDOS DE PROGRAMACION

- RUN
- STOP
- CONT
- END
- TRON
- TROFF
- FOR/NEXT
- GOTO
- IF.. THEN.. ELSE
- IF.. GOTO.. ELSE
- GOSUB
- RETURN
- ON... GOTO
- ON... GOSUB

RUN <nº línea>
Ejecuta el programa a partir del numero de lineas especificado

●<nº linea> numero de linea donde empezara la ejecucion del programa Si no se especifica, ejecutara el programa desde el principio.

RUN	: Ejecuta todo el programa.
RUN 120	Ejecuta el programa a partir de la linea 120

STOP
Detiene la ejecucion de un programa y se pone a la espera de nuevas órdenes

Cuando el ordenador encuentra un **STOP,** imprimira el siguiente mensaje:

BREAK IN • (• es el num de linea donde se encuentra el STOP)

A diferencia del comando **END, STOP** no cierra ficheros

La ejecucion del programa puede continuar utilizando el comando **CONT.** (ver **CONT.**).

CONT
Continua la ejecucion de un programa después de un **BREAK** o **STOP.**
No podrá utilizarse en los siguientes casos

1 Cuando la interrupcion sea debida a un error (ver código de Errores)
2 Cuando se efectue cualquier modificacion dentro del programa.
3 Cuando se incorpore una nueva linea en el programa

Al utilizar CONT en cualquiera de estos casos se imprimirá el mensaje de error: «Can't continue».

END
Termina la ejecucion de un programa cierra todos los ficheros y se pone a la espera de nuevas ordenes Es opcional

TRON
Visualiza en pantalla los numeros de linea por los que pasa el programa durante su ejecucion, facilitando asi el descubrimiento y la correccion de errores

Puede introducirse como instruccion directa antes de la ejecución del programa

TROFF
Anula la funcion de **TRON.**

FOR <variable> <valor inicial> **TO** <valor final> **STEP** <incremento>

Se utiliza conjuntamente con la sentencia **NEXT** para establecer una seccion de programa que se repita un numero dado de veces (Bucle **FOR NEXT**).

- <variable> Debe ser numerica. Inicialmente tendrá asignado el <valor inicial> incrementándose, segun el valor especificado en <incremento> hasta alcanzar el <valor final>.

- <valor inicial> Valor inicial que tendra la <variable>. Puede ser un numero o una expresión matematica en la que aparezcan valores numericos y/o variables relacionados por operadores.

- <valor final> Valor final que tomara la <variable>

- <incrementos> Define el valor del paso o salto que, sucesivamente, incrementara o decrementará el valor de la <variable> según sea positivo o negativo.
  Si no se especifica, el incremento será 1 **(STEP1)**

NEXT <variable.
Es la ultima instruccion de un bucle **FOR NEXT.**
- <variable> Nombre de variable que identifica el bucle **FOR NEXT.** Debe corresponder con la variable especificada en **FOR.**

NEXT, puede adoptar 3 formas distintas:

NEXT	cierra el ultimo bucle en caso de que hubiese varios
NEXT A	cierra el bucle identificado con la variable **A.**
NEXT A,B	cierra primero el bucle identificador con la variable **A** y luego el identificado con la variable **B**

1 Imprime el cuadrado de los 100 primeros numeros

```
10 FOR I = 1 TO 100
20 PRINT I∧2
30 NEXT I
```

2 Imprime todos los numeros pares de 0 a 100 en modo decreciente

```
10 FOR I = 100 TO 2 STEP - 2
20 PRINT I
30 NEXT I
```

3 Lee una lista de 10 numeros con **READ/DATA** y los imprime en pantalla.

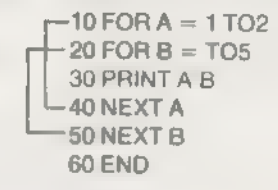
```
10 FOR X = 1 TO 10
20 READ A (X)
30 PRINT A (X)
40 NEXTX
60 DATA 100, 45, 320, 65, 43, 22, 4, 8, 9, 2.
```

Dentro de un mismo programa pueden utilizarse bucles **FOR/NEXT** anidados, ejecutándose unos dentro de otros (no pueden entrelazarse).

Bucles anidados correctos

```
10 FOR A = 1 TO2
20 FOR B = 1 TO5
30 PRINT A B
40 NEXT B
50 NEXT A
60 END
```

Bucles anidados incorrectos



GOTO <num. linea>
Salto incondicional al numero de linea especificado

- <num linea> numero de linea en el que continuara la ejecucion del programa.

Es caso de especificar un numero de linea que no existe en el programa se imprimira el siguiente mensaje de error "Undefined line number"

```
10 GOTO 40
20 END
40 PRINT "VENGO DE LA LINEA 10"
50 GOTO 20
```

IF <1ª expresión> **THEN** <2ª expresión> **ELSE** <3ª expresión>

IF es un «si» condicional. Es decir, si la <1ª expresión> se cumple, entonces **(THEN)** ejecuta la <2ª expresión>, que puede ser un número de línea, con lo cual pasará el control del programa a dicho número de línea, o bien una instrucción concreta.
Si la <1ª expresión> no se cumple, pasará a ejecutar la <3ª expresión>, que igualmente puede ser un número de línea o una instrucción concreta.

```
10 IF A = B THEN 80 ELSE 140
Si A es igual a B, entonces (THEN) ve a la línea 80
De lo contrario (ELSE), ve a la 140.
```

```
10 IF A = B THEN PRINT "A = B" ELSE PRINT "A<>B"
Si A es igual a B, entonces (THEN) imprime "A = B"
De lo contrario (ELSE), imprime "A<>B"
```

Si no se utiliza **ELSE** y la <1ª expresión> no se cumple, continuará la ejecución en la siguiente línea del programa.

```
10 IF A = B THEN PRINT "A = B"
20 PRINT "A<>B"
Si A es igual a B imprimirá "A = B".
De lo contrario, imprimirá "A<>B".
```

IF <1ª expresión> **GOTO** <num línea> **ELSE** <2ª expresión>

Es similar a **IF THEN ELSE** con la diferencia que **GOTO** irá siempre acompañado de un número de línea.
(Ver **IF THEN ELSE**)

```
10 IF A   20 GOTO 60 ELSE C   0
```

GOSUB <num. linea>
Desvia la ejecucion del programa hacia una subrutina

- <num linea> linea de inicio de la subrutina

Una subrutina, es un pequeño programa que realiza una funcion concreta Si durante un programa se necesita realizar una determinada funcion varias veces, en lugar de escribir las lineas de programa que realizan dicha funcion tantas veces como se precise ejecutarla, se escribe una sola vez convirtiendola en una subrutina

A lo largo del programa, el ordenador se dirigira a dicha subrutina tantas veces como se le indique retornando a la ejecución normal del programa principal una vez haya realizado la subrutina en cuestión

RETURN
Retorno, desde una subrutina, al programa principal Es la ultima instrucción de una subrutina, continuando la ejecucion del programa en la instrucción siguiente a la que fue llamada mediante **GOSUB.**

```
10 PRINT "EJEMPLO SUBRUTINA"
20 GOSUB 500
30 PRINT "VUELVO DE LA SUBRUTINA"
35 END
500 PRINT "ESTOY EN LA SUBRUTINA"
510 RETURN
```

ON <expresion> **GOTO** <lista nº linea>

Salta a un numero de linea del programa, determinado por el valor de <expresión>

- <expresion> variable o expresion de cuyo valor depende el numero de linea donde se efectuara el GOTO

- <lista nº linea> numero de lineas correspondientes a los saltos que efectuara **GOTO** segun el valor de <expresion> Por

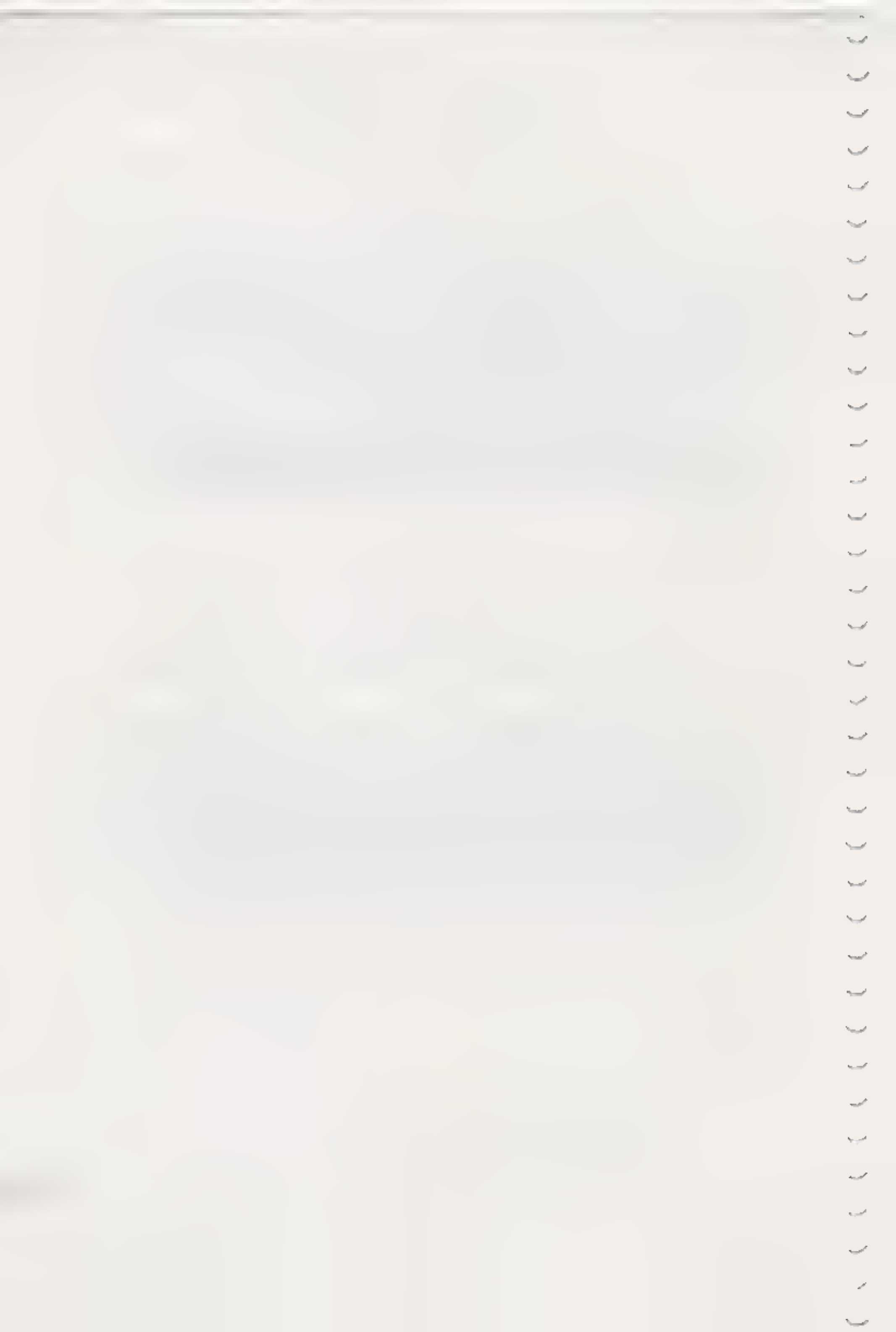
ejemplo, si el valor de <expresión> es 3, se hará un **GOTO** con el 3er numero de la lista
Los numeros de linea deberán estar separados por coma

```
ON A GOTO 70, 100, 190, 600
● Si A - 0 salta a la instruccion siguiente a ON GOTO
● Si A = 1, salta a la línea 70.
● Si A = 2, salta a la línea 100
● Si A = 3, salta a la línea 190
● Si A = 4, salta a la línea 600
● Si A >   5 salta a la siguiente instruccion a ON GOTO
```

ON <expresión> **GOSUB** <lista nº linea>

Es similar a **ON GOTO,** con la diferencia de que los saltos se efectuaran a las subrutinas. cuyas lineas de inicio son las indicadas en <lista nº línea>

```
ON A*5 GOSUB 100, 300
● Si A*5   0 Salta a la instrucción siguiente a ON GOSUB
● Si A*5   1, Salta a la subrutina que empieza en la linea 100
● Si A*5 · 2, Salta a la subrutina que empieza en la linea 300
● Si A*5   3, Salta a la instruccion siguiente a ON GOSUB
```



3. DEFINICION E INICIALIZACION

- CLEAR
- DIM
- ERASE
- DEFINT
- DEFSNG
- DEFDBL
- DEFSTR
- DEF USR
- USR
- KEY
- TIME

CLEAR
Asigna a todas las variables numéricas el valor cero y borra todos los datos acumulados en las variables alfanuméricas

Usado como comando, seguido de un número además de realizar las funciones descritas, en el párrafo anterior, reserva el número especificado en **BYTES** para las cadenas relativas a las variables alfanuméricas

CLEAR	Asigna a todas las variables el valor cero
CLEAR 500	Asigna a todas las variables el valor cero, y reserva 500 Bytes para las cadenas relativas a las variables alfanuméricas

NOTA: Inicialmente, el ordenador reserva un espacio de 200 Bytes para cadenas alfanuméricas pudiendo ser ampliado mediante CLEAR si las necesidades del programa lo requieren

DIM variable, <(dimensión)>

Dimensiona una variable subindicada (numerica o alfanumerica), determinando el numero maximo de elementos que podra contener

- <variable>: variable a dimensionar.
- <dimensión>: dimension de la variable

(ver apartado 1 5 2 variables subindicadas)

Las variables subindicadas podran tener uno o mas subindices y por consiguiente, capaces de albergar simultaneamente varios valores

```
DIM A (10)
DIM A (2,5)
DIM A (2,3,2)
DIM A (2,2,3,4,...)
```

ERASE <lista de variables>

Borra las variables dimensionadas especificadas en <lista de variables>, con el fin de poder dimensionarlas de nuevo sin que se produzca el error: "Redimensioned Array".

- <lista de variables> variables dimensionadas a borrar

```
10 DIM A (15)
20 FOR I = 0TO 15
30 A (I) = I + 1
40 NEXT I
50 ERASE A
60 DIM A (20)
```

Sin la instruccion de la linea 50, **ERASE A,** no podemos dimensionar de nuevo la variable A en la linea 60, pues se producira el error "Redimen-

sioned Array' ya que una misma variable no puede dimensionarse dos veces

DEFINT <lista de variable>
Define las variables especificadas como enteras (sin parte decimal)

- <lista de variables> variables que van a ser definidas como enteras Deben ir separadas por una coma

```
10 DEFINT A, B, X
20 A = 7.18  B = 16 8  X = 1.6
30 PRINT A, B, X
   (RUN)
7   16   1
```

DEFSNG <lista de variables>
Define las variables especificadas, en simple precision (Se representaran 6 digitos como máximo).

- <lista de variables> variables que van a ser definidas en simple precision Deben ir separadas por una coma

```
10  DEFSNG  A, B, X
20  A = 128345.6342
30  B = 3256 83716
40  X = 12 684372
50  PRINT A,B,X
        (RUN)
128345   3256 83      12 6843
```

DEFDBL <lista de variables>
Define las variables especificadas, en doble precisión (Se representaran 14 digitos como máximo).

- <lista de variables> variables que van a ser definidas en doble precisión. Deben ir separadas por coma.

```
10 DEFDBL  A, B
20 A = 30/4.3:B = 45/87
30 B = PRINT A,B
(RUN)
6.9767441860465       0.51724137931034
```

DEFSTR <lista de variables>
Define las variables especificadas como alfanuméricas

● <lista de variables> variables que van a ser definidas como alfanuméricas. Deben ir separadas por coma

```
10 DEFSTR A,B
20 A = "BASIC"
30 B = "MSX"
40 PRINT A +        + B
   (RUN)
   BASIC MSX
```

NOTA: En todos los comandos de declaración de caracteres (DEFINT, DEFSNG, DEFDBL, DEFSTR), puede especificarse también un rango de variables a declarar. Por ejemplo: DEFINT A-D, define las variables A,B,C,D, de tipo entero.

DEFFN <nombre><(lista de parametros)> <definición de la función>

Permite definir una función por el usuario

●<nombre> Puede ser el nombre de cualquier variable. Este <nombre> precedido de **FN**, será el nombre de la función

●<(lista de parametros> Estará compuesta por todas las variables que existan en la <definición de la función> y serán sustituidas al llamar a dicha función. Las variables deben estar separadas por coma

● <definicion de la funcion> Expresion matematica que realiza las operaciones de la funcion definida, estando limitada a una linea (255 caracteres).
Las variables utilizadas en esta expresion unicamente sirven para definir la funcion, no influyendo las variables del programa que tengan el mismo nombre. Si en posteriores llamadas a esta funcion, se omite el valor de una variable sera tomado su ultimo valor.
La llamada a la funcion se realizara utilizando la siguiente estructura:
FN <nombre> <(valores de las variables)>

```
10 DEFFNR (A,B) = (A*B)/2
.
60 A = FNR (4,5)
70 PRINT A
   (RUN)
   10
```

Si se realiza la llamada de una funcion antes de que esta haya sido definida, se producira el error «Undefined user function».
DEFFN no puede utilizarse como instruccion directa.

DEFUSR <numero> = <dirección>
Define la direccion de inicio de una subrutina en lenguaje maquina.

- <numero> numero de la subrutina **USR**. Debe ser un entero entre 0 y 9. Si se omite, se asignara por defecto el valor 0 (**DEFUSR** 0).
- <direccion> Direccion de inicio de la subrutina **USR** especificada.

```
DEFUSR 0 = 53248
```

USR <número> <(x)>

Ofrece el resultado obtenido al ejecutar una rutina en lenguaje maquina que empieza en la direccion definida por DEFUSR.

- <numero> Identifica la subrutina con un numero determinado que debe ser un entero entre 0 y 9. Si se omite, se asignara por defecto el valor 0. (**USR**0).

● <(x)> Parametro que sera utilizado por la subrutina en lenguaje ensamblador.

KEY <número>, <"expresión">
Define las teclas de función F1 a F10

●<numero> numero de la tecla de funcion a definir Debe ser un entero entre 1 y 10

●<«expresion»> sentencia comando función operación, etc , que se va a definir.
Puede tener como maximo 15 caracteres.

```
KEY 1,  BASIC MSX
KEY 3,  SQR
KEY 4, "3 1416
A$   "ORDENADOR   KEY 5, A$  KEY 7   LPRINT
```

Para utilizar el contenido de las teclas programadas, ejecute KEY LIST (ver KEY LIST para más detalles)

TIME
Temporizador interno del sistema Al inicializar el sistema se pone a cero y va incrementandose en 1 cada vez que VDP genera interrupción (60 veces por segundo) Dividiendo el valor de **TIME** por 60 obtendremos un contador de segundos.

TIME es una variable especial del sistema Automaticamente toma el valor 0 y va incrementandolo, aunque tambien podemos asignarle un valor inicial por programa (**TIME** <valor>)

```
10 LOCATE 17 2  PRINT INT (TIME/60)
20 GOTO 10
```

4. COMANDOS Y FUNCIONES DE I/O

- READ
- DATA
- RESTORE
- INPUT
- LINE INPUT
- LET
- MID$ (   ) = Y$
- PRINT
- PRINT USING
- LPRINT
- LPRINT USING
- LPOS
- SWAP
- INKEY$
- INPUT$
- FRE
- POKE
- PEEK
- VPOKE
- VPEEK
- INP
- OUT
- WAIT
- STICK
- STRIG

READ <lista de variables>
Lee los datos correspondientes a una sentencia **DATA**, asignándolos a las variables especificadas en <lista de variables>

<lista de variables> Pueden ser numericas o alfanumericas y la unica condicion que deben cumplir es que esten separadas por coma

```
READ A$ B C
```

READ se utiliza conjuntamente con **DATA**. Cuando el Ordenador encuentra una sentencia **READ**, buscara el **DATA** correspondiente en cualquier lugar del programa.

```
10 READ A$, B
:
500 DATA PEDRO 23
```

Cuando el ordenador llega a la linea 10, realizara una operacion de lectura en el **DATA** de la linea 500, asignando a la variable **A$** el valor PEDRO y a B el valor 23.

DATA <lista de datos>
Constituye un archivo de datos dentro del programa, que en cualquier momento pueden ser leidos por READ.

<lista de datos> Pueden ser numeros caracteres o ambas cosas y deben estar separados por coma Estos valores seran asignados sucesivamente a las variables de **READ**.

El numero de datos contenido en <lista de datos> nunca puede ser menor que el numero de variables contenida en <lista de variables> de la sentencia **READ**. De otra forma, se originaria e error "Out of Data".

Si dentro de <lista de datos> queremos introducir un dato que contenga signos especiales como coma ( ) dos puntos ( ) etc , estos deberan ir entre comillas Las comillas no pueden utilizarse como dato dentro de **DATA**.

```
10 REM$ EJEMPLO READ/DATA$
20 FOR I = 1 TO 2
30 READ A$, B$, C$
50 NEXT I
60 DATA HIT, BIT, 55, SONY
70 DATA BASIC, MSX
```

En la primera operacion de lectura (I – 1), los valores asignados a A$, B$ y C$ son respectivamente HIT, BIT, 55

En la segunda lectura (I - 2), los valores asignados serán SONY, BASIC, MSX.

RESTORE <núm. línea>
Indica el numero de linea donde se encuentra el **DATA** que contiene los datos a leer por **READ.**

Si no se indica <num linea>, se reestablecerá el orden de lectura, colocando el puntero en el primer dato de la primera linea DATA del programa.

```
10 READ A$, B
:
50 RESTORE
60 READ A$,B
:
100 DATA PEDRO, 23
```

Sin la instrucción **RESTORE** de la linea 50 se produciria el error OUT OF DATA , ya que el **READ** de la linea 60, intentaria leer los datos posteriores a PEDRO 23 que no existen De esta forma restablecemos el orden de lectura, colocando el puntero en el primer dato (PEDRO), por lo que las instrucciones de las lineas 10 y 60 leerán los mismos datos.

```
10  READ A$, B

50  RESTORE 520
60  READ C,D,E,F



500  DATA PEDRO, 23
510  DATA 100, 200
520  DATA 400, 350, 600
530  DATA 800, 900
```

La línea 10, lee los datos PEDRO, 23.
La línea 50 hace que el próximo **READ** se efectúe a partir de la línea 520, originándose los siguientes valores C = 400, D = 350, E = 600, F = 800.

Sin la instrucción de la línea 50 (**RESTORE** 520), los valores se habrían asignado de la forma:
C = 100, D = 200, E = 400, F = 350.

INPUT <lista de variables>
Asigna un valor, introducido desde el teclado, a una variable

- <lista de variables> Variables (numéricas o alfanuméricas) a las que serán asignados los valores introducidos desde el teclado.

Si durante la ejecución del programa éste se encuentra con el comando **INPUT**, la ejecución se detendrá, apareciendo en pantalla un interrogante (?), y permanecerá a la espera hasta que introduzcamos un dato desde el teclado, que será asignado a la variable asociada al comando **INPUT**.

```
INPUT A
INPUT A$
```

El comando **INPUT** nos permitirá, si así lo deseamos, incluir un mensaje, que deberá ser escrito entre comillas, y que en el momento de ser ejecutado se imprimirá en la pantalla.

```
20 INPUT "ESCRIBE UN NUMERO " X
30 PRINT ' EL CUADRADO DE , X "ES ', X*X
```

Al ejecutar el programa anterior, el ordenador nos pedira el valor de X

```
ESCRIBE UN NUMERO. ?
```

Una vez introducido se lo asignara a la variable X y continuara con la ejecucion del programa imprimiendo el cuadrado del valor introducido

Con un solo comando **INPUT**, pueden asignarse varios valores a distintas variables, siempre que esten separadas por coma **INPUT** A,B C
Asi mismo, al introducir los datos desde el teclado, deberemos separarlos por coma.

```
10 INPUT "INTRODUZCA LA FECHA" D,M A
(RUN)
? 31 12 84
```

Si en un comando **INPUT**, que tiene asociada una variable numérica, introducimos un valor alfanumerico se imprimira el mensaje de error «Redo from start», permaneciendo de nuevo a la espera del dato (Recuerde que despues de teclear el valor de entrada debe pulsar la tecla RETURN)

```
10 INPUT "A y B"; A,B
20 PRINT A*B
(RUN)
A y B? 10, PE
REDO FROM START
A y B? 10, 20
200
OK
```

```
10 INPUT "SU NOMBRE"; NO$
20 PRINT "HOLA,"; NO$
```

```
10 INPUT "NOMBRE Y EDAD: " N$,E
20 PRINT, N$; "TIENE "; E; "AÑOS "
```

LINE INPUT <variable>
Asigna una cadena de caracteres, introducidos desde el teclado, a una variable alfanumérica. El número máximo de caracteres de la cadena es 254.

- <variable> variable alfanumérica a la que se asignará la cadena de caracteres introducida desde el teclado.

LINE INPUT es similar a INPUT, teniendo en cuenta las siguientes diferencias:

- No imprime el interrogante.
- Sólo puede utilizarse con variables alfanuméricas.
- Sólo puede utilizarse con una variable.

```
10 LINE INPUT "NOMBRE, APELLIDOS Y
DIRECCION"; A$
     (RUN)
NOMBRE, APELLIDOS Y DIRECCION
```

Todos los caracteres, introducidos hasta que se pulse RETURN, serán asignados a la variable A$.

LET <variable> = <expresión>
Asigna un valor a una variable.

- <variables> variable a la que se asignará el valor <expresión>. Puede ser numérica o alfanumérica.

- <expresión> valor que se asignará a la variable especificada.

```
LET A = 50
LET A$ = "NOMBRE"
LET C = A+B
```

LET es opcional, ya que directamente puede asignarse una expresión a una variable mediante el signo «igual» (=), sin utilizar dicho comando.

Así, las siguientes instrucciones realizan la misma función:

```
10 LET A = 100
10 A = 100
```

**MID$** <(X$, N, M)> = Y$
Extrae M caracteres de la variable X$, a partir de la posición N, sustituyendolos por M caracteres de la variable Y$

Si se omite M, se colocarán todos los caracteres de Y$
Y$ no debe ser mayor que X$

```
10 A$ = "ORDENADOR"
20 B$ = "SONY"
30 MID$ (A$, 3,3) = B$
40 PRINT A$
(RUN)
ORSONADOR
```

PRINT <expresión>
Imprime en pantalla la <expresión>

- <expresión>: Puede ser de varios tipos.

1. Valores numéricos:
   PRINT 50

2. Variables numéricas cuyos valores han sido asignados previamente:
   10 A = 3
   20 PRINT A
   Imprime en pantalla: 3

3. Variables de cadena cuyos valores han sido asignados previamente
   10 A$ = "MSX"
   20 PRINT A$
   Imprime en pantalla: MSX

4. Expresiones que contienen operaciones aritmeticas o logicas
   10 A = 10
   20 PRINT 3*2+A
   Imprime en pantalla 26 (resultado de 3*2+10)

5. Cadenas encerradas entre comillas.
   10 PRINT "BASIC MSX"
   Imprime en pantalla: BASIC MSX

Las expresiones separadas por coma se imprimiran en dos campos distintos.

```
10 A = 5: B = 10
20 PRINT A,B
(RUN)
5    10
```

Las expresiones separadas por punto y coma (;) se imprimiran a continuacion de la anterior, teniendo en cuenta que las expresiones numericas estaran separadas por dos espacios

Si al final de la ultima expresión de un comando **PRINT** hay un punto y coma la próxima sentencia **PRINT** se imprimirá a continuación de ésta

Si se utiliza **PRINT** sin ninguna expresion se imprimira una linea en blanco.

El simbolo de interrogación (?) puede utilizarse en lugar de la palabra **PRINT**

```
PRINT 10
? 10
```

Ambas expresiones realizan la misma funcion (imprimen en pantalla 10)

PRINT USING <Formato>, <expresión>
- Imprime en pantalla la <expresion> utilizando el <formato> especificado.
- <Formato> caracteres que definen el formato de impresión
- <expresion> variables o expresiones que van a ser impresas con el formato especializado Deberan estar separadas por coma

Cuando se usa PRINT USING para imprimir variables alfanumericas, pueden utilizarse los siguientes caracteres para su formateado
1 ' indica que sólo se imprimirá el primer caracter de la variable alfanumerica.

2 " + " el signo de adición indica que el signo del numero a escribir se colocará delante o detrás del mismo.

```
PRINT USING "+ # # #. # #"; 1.26, -1.25
+1.25   -1.25.
Ok
PRINT USING "# # # . # # +"; 1.25,-1.25
     (RUN)
     1 25 + 1.25-
```

3 " - " el signo de sustracción al final del formato implica la colocación de dicho signo al final de los numeros negativos

```
PRINT USING "# # #. # #—"; 1.25,  -1.25
Ok
1 25    1.25-
```

4 "* *" un doble asterisco al principio del formato indica que aparecera el asterisco en los lugares en que sobre espacio, previos al digito Tambien indica dos o mas lugares para digitos

```
PRINT USING"** # # . #";1.25, -1.25
  *1.25      *-1 25
Ok
```

5 "£ £" aparecerá el simbolo de la libra inmediatamente a la izquierda del numero en cuestion Los £ £ ofrecen dos posiciones más de digitos una de las cuales es el signo de la libra La forma exponencial no puede ser usada con £ £.

```
PRINT USING "£ £ # # #"; 5680
£ 5680
Ok
```

```
A$ = "España"
Ok
PRINT USING "!"; A$
E
Ok
```

2 "\ n espacios \" indica que van a imprimirse 2+n caracteres de la variable alfanumérica.

```
A$ = "España"
Ok
PRINT USING "\   \"; A$
ESPAÑ
Ok
```

3 "& Indica que serán impresos todos los caracteres de la variable

```
A$ = "España"
Ok
PRINT USING "Me gusta mucho &"; A$
Me gusta mucho España
Ok
```

Cuando se usa **PRINT USING** para imprimir datos numéricos pueden utilizarse los siguientes caracteres para su formateado

1 "#" se usa para representar cada posición de dígito Si el número a ser impreso tiene menos dígitos que posiciones especificadas, el número será precedido por tantos espacios en blanco como # sobrantes

```
PRINT USING "##.## "; 10.983, 8.6, 764.3
    (RUN)
    10.98   8.60   764.30
```

6 "**£" *combina los efectos de los dos simbolos anteriores Los espacios en blanco serán llenados con asteriscos y el signo de la libra aparece antes del numero Los * * £ ofrecen tres posiciones más de digitos, uno de los cuales es £.*

```
PRINT USING "* * £#.# #"; 12.35
* £ 12.35
Ok
```

7 ," una coma situada a la izquierda del punto decimal en la secuencia implica que una vez ejecutada la sentencia, aparecerá una coma situada tres digitos a la izquierda del decimal Una coma situada al final de la secuencia, se imprimirá al final del numero

```
PRINT USING "# # # #, # #"; 1234 5
1,234.50
Ok

PRINT USING "# # # #.# #,"; 1234.5
1234 50,
Ok
```

8. "- - ^ ^   convierten el numero dado en exponencial

```
PRINT USING "# #.# # ^^^^"; 234.56
235 E+02

Ok
PRINT USING "#.#  # # ^^^  -  , -12 34
1.23E+01-
Ok
PRINT USING "+#.# #  ^^^^"; 12.34, -12.34
+1.23E+01 - 1.23E+01
Ok
```

9. ' %" si el numero a imprimir tiene más digitos que los previstos en la instruccion se imprimirá el simbolo % junto al numero Lo mismo

sucede si la operacion de redondeo causa que el numero contenga más dígitos de los previstos.

```
PRINT USING "# #.# #"; 123 45
% 123 45
Ok
PRINT USING ".# #",.998
% 100
OK
```

Si el numero de digitos excede de 24 aparece el mensaje de error «Illegal function call».

LPRINT
Imprime en impresora Sigue las mismas normas que **PRINT.** (ver **PRINT** para más detalles).

LPRINT USING
Imprime en impresora con un formato especifico Sigue las mismas normas que **PRINT USING.** (ver **PRINT USING** para mas detalles)

LPOS
Realiza la función de **POS** referida a la impresora (Ver comando **POS** para más detalles)

SWAP <X>, <Y>
Intercambia los valores de las variables especificadas Ambas variables deben ser del mismo tipo (enteras, simple precision,) de lo contrario se producirá el error: "Type mismatch."

- <X> variable que tomara el valor de la variable <Y>
- <Y> variable que tomara el valor de la variable <X>

```
10 A = 5 B = 10
20 PRINT "A = "; A, B ", B
30 SWAP A,B
```

```
40 PRINT "A = "; A, "B    ", B
(RUN)
A = 5      B = 10
A = 10      B = 5
```

INKEY $

Devuelve una cadena de un solo caracter definido por la pulsación de una determinada tecla Si no se pulsa ninguna tecla durante el periodo de observacion devuelve una cadena nula

Debido a la corta duracion de los ciclos de observacion, **INKEY $** se situa dentro de alguna clase de bucle para que sea observado continuamente.

```
10 A$ = INKEY$
20 IF A$ = "  " GOTO 10
```

INPUT$ (x)

Detiene el programa hasta que se hayan introducido x caracteres Los caracteres introducidos no seran visualizados en pantalla Pulsando **CTRL-C ó CTRL-STOP** se anulara **INPUT$ (X).**

```
10 X$    INPUT$ (25)
```

FRE <(expresión)>

Determina el numero de **BYTES** libres en memoria **RAM.**

- <(expresion)> Puede ser cualquier caracter alfanumerico no influyendo en el resultado Si el caracter va entre comillas, nos indicara el numero de **BYTES** reservados para las cadenas relativas a variables alfanumericas (Ver **CLEAR** para más detalles)

```
PRINT FRE (X)
PRINT FRE ("Z")
```

POKE <dirección>, <dato>
Escribe un dato en la posición de memoria especificada

- <direccion> Dirección de memoria donde se escribirá el dato Debe estar comprendida entre -32768 y 65535
Si el valor es negativo, la direccion real será 65 536 menos el valor especificado. (-1, será calculado como 65.536-1 = 65.535)
(ver Mapa de memoria)

- <dato> Dato **(BYTE)** que va a ser escrito en la direccion de memoria especificada Debe ser un numero entre 0 y 255

```
POKE 63.522, 65
```

PEEK <(dirección)>
Es la funcion complementaria de **POKE.** Lee el dato (numero entero entre 0 y 255) almacenado en la direccion de memoria especificada

- <(dirección)> Dirección de memoria donde se encuentra el dato que deseamos leer.

Para visualizar el dato leido, utilizaremos PRINT

```
PRINT PEEK (63 522)
```

VPOKE <dirección>, <dato>
Almacena un dato en la direccion especificada de la V RAM (memoria de pantalla)

- <direccion> Debe ser un numero entero entre 0 y 16 383 (ver Mapa de memoria).

- <dato> Dato **(BYTE)** que va a ser almacenado en la direccion de memoria especificada Debe ser un entero entre 0 y 255

```
VPOKE 14500, 65
```

VPEEK <(dirección)>
Es la funcion complementaria de **VPOKE.** Lee el dato (numero entero

entre 0 y 255) almacenado en la direccion especificada de la **VRAM** (memoria de pantalla)

● <direccion> Debe ser un numero entero entre 0 y 16 383

```
PRINT VPEEK (14500)
```

INP <port>
Lee el **BYTE** almacenado en el port especificado

● <port> numero del port donde va a efectuarse la lectura Debe ser un entero entre 0 y 255

INP es la función complementaria de **OUT**

```
INP (&H92)
```

OUT <port>, <valor>
Escribe un BYTE en el port de salida especificado

● <port> numero del port donde va a efectuarse la operación de escritura. Debe ser un entero entre 0 y 255.
● <valor> Dato (BYTE) que va a ser transmitido Debe ser un entero entre 0 y 255.

```
OUT &H90 3
```

WAIT <port>, Y, Z
Efectua una pausa hasta que el dato del port de entrada cambie a un valor determinado

● <port> numero del port de entrada utilizado

La secuencia de operaciones que implica la ejecucion de esta instrucción, son las siguientes.

1. Lée el dato en <port>
2. Opera en forma OR exclusiva (EX-OR) el dato leido con el valor Z

3 Efectúa el producto lógico AND entre el resultado de la operación EX-OR y el valor Y.
Si el resultado es 0 realiza de nuevo todo el proceso En caso contrario prosigue la ejecución del programa a partir de la instrucción que sigue a **WAIT**

STICK <(N)>
Determina el movimiento realizado por el Joystick dando un valor numérico según la siguiente codificación

Movimiento	STICK (N)
• Centro	0
• Ascendente	1
• Diagonal derecha ascendente	2
• Derecha	3
• Diagonal derecha descendente	4
• Descendente	5
• Diagonal izquierda descendente	6
• Izquierda	7
• Diagonal izquierda ascendente	8
•<(N)>: indica el Joystick utilizado. N = 0: teclado N 1 Joystick 1 N 2 Joystick 2	

Conecte el Joystick 1 y mueva la palanca para visualizar los distintos valores tomados según el movimiento realizado

```
10 LOCATE 5,3
20 PRINT STICK (1)
30 GOTO 10
```

STRIG <(N)>
Toma el valor -1 al pulsar el botón de disparo del Joystick Si no se pulsa, toma el valor 0

•<(N)>: Indica el Joystick utilizado
N 0 teclado (botón de disparo barra espaciadora)
N = 1: Joystick 1
N = 2: Joystick 2

5. INTERRUPCIONES

- **ON KEY GOSUB**
- **KEY (N) ON/OFF/STOP**
- **ON STRIG GOSUB**
- **STRIG (N) ON/OFF/STOP**
- **ON STRIG GOSUB**
- **STOP ON/OFF/STOP**
- **ON SPRITE GOSUB**
- **SPRITE ON/OFF/STOP**
- **ON INTERVAL - < > GOSUB**
- **INTERVAL ON/OFF/STOP**

ON KEY GOSUB <nº de linea>
Interrumpe el programa principal al pulsar una determinada tecla de funcion (F1-F10) pasando a realizar la subrutina de interrupcion indicada en <nº de linea>.

- <nº de linea> numeros de lineas iniciales de las subrutinas

ON KEY GOSUB 100, 200, 350, 760,...

100 es la linea de inicio de la subrutina de interrupcion correspondiente a la tecla de función F1. **(KEY (1)).**

200 es la línea de inicio de la subrutina de interrupción correspondiente a la tecla de función F2 **(KEY (2))**.

.

.

.

.

Al producirse una interrupción se ejecutará automáticamente un KEY (n) STOP (ver comando **KEY ON/OFF/STOP**)

Si dentro de la subrutina de interrupción no se ha especificado KEY-(n)OFF, después de ser ejecutado el **RETURN**, (retorno de la subrutina al programa principal) se ejecutará también un KEY(n)ON, quedando de nuevo habilitada la interrupción para esta tecla

Si se produce una interrupción de error (debida a un comando ON ERROR), se inhibirán automáticamente todas las interrupciones (incluidas **ERROR, STRIG, STOP, SPRITE, INTERVAL, KEY**)

KEY ON/OFF/STOP

Se utilizan para habilitar/inhibir interrupciones con las teclas de función Fn. (KEY(n)).

- KEY(n)ON Permite interrupción con la tecla de función Fn (KEY(n)

```
 10 KEY (1) ON
 20 ON KEY GOSUB 200
 30 REM PROGRAMA PRINCIPAL
 40 FOR X = 1 TO 100
 50 PRINT X
 60 NEXT X
 70 CLS
 80 GOTO 30
200 REM SUBRUTINA DE INTERRUPCION
210 CLS.
230 PRINT "HAS PULSADO F1"
230 FOR I=1 TO 100
```

```
240 LOCATE 0,10: PRINT I
250 NEXT I
260 CLS
270 RETURN
```

En la linea 10 indicamos al ordenador que «permitimos» interrumpir el programa pulsado F1.

En la linea 20, le indicamos que si durante el transcurso del programa principal se pulsa la tecla F1, pase a ejecutar la subrutina que empieza en la linea 200

El programa principal (10-80) se ejecutara con toda normalidad hasta que pulsemos F1, momento en que se detendra, pasando a ejecutar la subrutina de interrupcion correspondiente Una vez finalizada la ejecucion de esta subrutina el programa seguira su curso a partir del punto donde fue interrumpido, pudiendo ser interrumpido de nuevo del mismo modo anterior.

KEY(n)OFF Inhibe la interrupcion de la tecla de funcion Fn (KEY(n)) Si en el ejemplo anterior sustituimos la linea 10 por 10 KEY (1) OFF aunque pulsemos F1, no se interrumpira el programa siguiendo éste su curso normal

KEY(n)STOP Si previamente a KEY(n)STOP se habilito la interrupción mediante KEY(n)ON al pulsar la tecla Fn no se producirá interrupción, pero memorizara que se pulsó dicha tecla y pasará a ejecutar la subrutina correspondiente despues de encontrar la autorización de interrupcion KEY(n)ON

```
10 KEY(1)ON
20 PRINT "BASIC MSX"
30 KEY(1) STOP
40 ON KEY GOSUB 500



100 PRINT "FIN = F1"
```

```
110 KEY(1)ON



500 PRINT "FIN"
510 RETURN
```

Aunque pulsemos F1 el programa no sera interrumpido hasta que aparezca e mensaje que imprime la linea 100 «FIN F1», ya que es en la linea 110 donde se habilito de nuevo la interrupcion para F1

ON STRIG GOSUB <nº de lineas>
Interrumpe el programa principal cuando se pulsa el boton de disparo de los Joysticks o la barra espaciadora del teclado
Su funcionamiento es similar a **ON KEY GOSUB.** (Ver **ON KEY GOSUB** para más detalles).

STRIG(n) ON/OFF/STOP
Se utilizan para habilitar/anular interrupciones con el boton de disparo de los Joysticks o con la barra espaciadora del teclado

(n) debe ser un número entero entre 0 y 4.

- n = 0 la barra espaciadora es utilizada como botón de disparo
- n = 1 o 3 se utiliza el boton de disparo de Joystick 1
- n = 2 ó 4 se utiliza el boton de disparo del Joystick 2

Su funcionamiento es similar a **KEY ON/OFF/STOP** (Ver **KEY ON/OFF/STOP** para más detalles).

ON STOP GOSUB <nº. de linea>

Interrumpe el programa principal al pulsar **CTRL-STOP,** pasando a realizar la subrutina de interrupcion cuya linea de inicio es la indicada en <nº. de linea>.

Utilize con precaucion este comando Por ejemplo, el siguiente programa no podrá detenerse La unica solución es hacer un RESET o desconectar el aparato.

```
10 ON STOP GOSUB 40
20 STOP ON
30 GOTO 30
40 RETURN
```

STOP ON/OFF/STOP

Se utilizan para habilitar/anular interrupciones mediante **CTRL-STOP.** Su funcionamiento es similar a **KEY ON/OFF/STOP.** (ver **KEY**/ON/OFF/STOP).

ON SPRITE GOSUB <nº línea>

Interrumpe el programa principal cuando se solapan dos **SPRITES** en la pantalla.
Su funcionamiento es similar a **ON KEY GOSUB** (ver **ON KEY GOSUB** para más detalles).

SPRITE ON/OFF/STOP

Se utilizan para habilitar/anular interrupciones cuando coinciden dos SPRITES.

Su funcionamiento es similar a **KEY ON/OFF/STOP.** (ver **KEY ON/OFF/STOP** para más detalles).

ON INTERVAL <tiempo> **GOSUB** < nº línea >

Interrumpe el programa principal cada periodo de tiempo especificado en <tiempo>/60

```
10 ON INTERVAL = 60 GOSUB 500
```

Al ejecutar esta línea cada segundo (60/60) se interrumpirá el programa principal para ejecutar la subrutina que comienza en la línea 500

Su funcionamiento es similar a **ON KEY GOSUB,** con la diferencia de que la interrupción se producirá automáticamente cada cierto tiempo, sin tener que pulsar ninguna tecla.

INTERVAL ON/OFF/STOP

Se utiliza para habilitar/anular interrupciones temporales con **INTERVAL**

Su funcionamiento es similar a **KEY ON/OFF/STOP.**

(Ver **KEY ON/OFF/STOP** para más detalles)



6. FUNCIONES NUMERICAS

- ABS
- INT
- FIX
- SGN
- CDBL
- CSNG
- CINT
- EXP
- LOG
- SQR
- SIN
- COS
- TAN
- ATN
- RND

ABS (X)
Da el valor absoluto de X El resultado siempre será positivo

```
10 PRINT ABS (-5)
(RUN)
5
```

INT (X)

Calcula el mayor entero menor o igual a X.

```
10 PRINT INT (2.56)
   (RUN)
    2

10 PRINT INT (—2.56)
   (RUN)
    —3
```

FIX (X)
Da la parte entera de X FIX (X) es equivalente a SGN (X) * INT (ABS (X))

La mayor diferencia entre **FIX** e **INT** es que cuando trabajan con numeros negativos (X< O) **FIX** da la parte entera por exceso e **INT** por defecto

```
10  PRINT  FIX  (—2.56)
   (RUN)
    —2
```

SGN (X)

Determina el signo de X:

Si X>0 : SGN (X) = 1
Si X<0 : SGN (X) = —1
Si X = 0 : SGN (X) = 0

CDBL (X)

Representa el numero X en doble precisión

```
10 A = 53 45567893141785
20 PRINT CDBL (A)
   (RUN)
 53 455678931418
```

CSNG (X)
Representa el numero X en simple precision

```
10  A = 45.987666666634
20 PRINT  CSNG  (A)
(RUN)
53.4457
```

CINT (X)
Representa el numero X como entero eliminando la parte decimal X debe estar comprendido entre —32768 y 32767 En caso contrario se producira error de Overflow (desbordamiento)

```
10 A = 45 98765
20 PRINT CINT (A)
(RUN)
45
```

EXP (X)

Calcula la expresion e ˙ × debe ser menor o igual a 145 06286085862 (e. es la base de los logaritmos naturales e 2.7182818284588)

```
10 PRINT EXP (2)
(RUN)
7 38905609893
```

LOG (X)
Calcula el logaritmo neperiano de X (logaritmo en base e)

Para obtener el logaritmo de X en otra base (base Y) debera realizar la siguiente operación:

LOG (X)/LOG (Y)   Logaritmo de X en base Y

```
1. logaritmo neperiano de 4:
10  PRINT  LOG  (4)
   (RUN)
 1 3862943611199

2. logaritmo de 1000 en base 10

10  PRINT  LOG  (1000)/LOG  (10)
   (RUN)
    3
```

SQR (X)

Calcula la raiz cuadrada de X X debe ser un numero mayor o igual a cero De lo contrario se producirá el error   Illegal function call

```
10  PRINT  SQR  (25)
   (RUN)
    5
```

SIN (X)
Da el valor de seno de X. (en radianes).
El cálculo se efectúa en doble precisión.

```
10  PRINT  SIN  (3.1416/2)
   (RUN)
 99999999999324
```

COS (X)

Da el valor de coseno de X. (en radianes).
El calculo se efectua en doble precision

```
10 PRINT COS (0)
   (RUN)
    1
```

TAN (X)

Da el valor de tangente de X (en radianes)
El cálculo se efectua en doble precision

```
10 PRINT TAN (0.1)
(RUN)
10033467208545
```

ATN (X)
Da el angulo (en radianes), cuya tangente es X El resultado estará comprendido entre $-\pi/2$ y $+\pi/2$ El calculo se efectua siempre en doble precisión.

```
10 PRINT ATN (0.3)
(RUN)
.29145679447789
```

RND (X)

Genera un número aleatorio entre 0 y 1.

El argumento de **RND**, X se utiliza para controlar la generacion del numero aleatorio Cada vez que se ejecute un programa que contenga **RND**, se generara la misma secuencia de numeros aleatorios Durante la ejecucion de un programa si X<0 se generará la misma secuencia anterior Si X 0 repetira el ultimo numero generado y si X >0, generará el proximo numero aleatorio de la secuencia

```
10 PRINT RND (1)*100
20 PRINT RND (—1)*100
30 PRINT RND (0)*100
40 PRINT RND (1)*100
(RUN)
59 521943994623
4.384820420821
4 384820420821
9 62486816692
```



7. CONVERSION DE CODIGOS

- **ASC**
- **CHR$**
- **BIN$**
- **OCT$**
- **HEX$**
- **VAL**
- **STR$**

ASC <(X$)>
Determina el codigo ASCII del caracter X$

```
10 A$ = "A"
20 PRINT ASC (A$)
30 PRINT ASC ("B")
   (RUN)
    65
    66
```

CHR $ (X)
Realiza la funcion inversa de ASC Determina el carácter que en código ASCII corresponde al número X.

```
10 PRINT CHR $ (65)
(RUN)
A
```

BIN$ (X)
Convierte el numero decimal X en un numero binario El valor de X debe estar comprendido entre —32768 y 65535

Si X es negativo, dara el valor resultante de restar a 65 535 el valor de X (65 535 —X)

```
10 PRINT BIN$ (67)
(RUN)
1000011
```

OCT$ (X)

Convierte el numero decimal X en un numero octal El valor de X, debe estar comprendido entre —32768 y 65535.

Si X es negativo dara el valor resultante de restar a 65 535 el valor de X (65535—X).

```
10 PRINT OCT$ (76)
(RUN)
114
```

HEX$ (X)

Convierte el numero decimal X en un numero hexadecimal El valor de X, debe estar comprendido entre —32768 y 65 535

Si X es negativo, dara el valor resultante de restar a 65 535 el valor de X (65535 —X)

```
10 PRINT HEX$ (94)
(RUN)
5E
```

VAL (X$)

Determina el valor numerico de los primeros caracteres numericos de X$

```
10 A$ = "125A43"
20 B$ = "15C"
30 PRINT VAL (A$)
40 PRINT VAL (A$) + VAL (B$)
50 PRINT VAL (A$ + B$)
   (RUN)
 125
 140
 12515
```

STR$ (X)

Realiza la función inversa de VAL (X$)

```
10 A$ = STR$ (22)
20 B$ = "BASIC"
30 PRINT A$ + B$
   (RUN)
   22 BASIC
```

8. CADENAS DE CARACTERES

- **RIGHT$**
- **LEFT$**
- **MID$**
- **STRING$ (N,X)**
- **STRING$ (N,X$)**
- **SPACE$**
- **INSTR**
- **LEN**

**RIGHT$**<(X$,N)>
Extrae N caracteres de la cadena X$ empezando por la derecha

Si N = LEN (X$), nos dará X$

```
10 A$ = "BASIC MSX"
20 PRINT RIGHT$ (A$ 6)
30 PRINT RIGHT$ (A$,3)
   (RUN)
   IC MSX
   MSX
```

**LEFT$** <(X$,N)>
Extrae N caracteres de la cadena X$ empezando por la izquierda

```
10 A$ = "BASIC MSX"
20 PRINT LEFT$ (A$,7)
30 B$ = LEFT$ (A$,3)
40 PRINT B$
   (RUN)
   BASIC M
   BAS
```

**MID$** <(X$, N, M)>
Extrae N caracteres de la cadena X$, empezando por la izquierda, a partir de la posición M

```
10 A$ = "ORDENADOR SONY"
20 B$ = MID$ (A$, 8, 5)
30 PRINT B$
40 PRINT MID$ ("MSX", 1,2)
   (RUN)
  NADOR SONY
  S
```

STRING <(N, X)>
Crea una cadena con N caracteres X en codigo ASCII

```
10 PRINT STRING $ (3,65)
   (RUN)
   AAA
```

NOTA: El ASCII de 65 es A.

**STRING$** <(N, X$)>
Crea una cadena con N caracteres X$ iguales

```
10 A$ = "A"
20 PRINT STRING$ (5,A$)
   (RUN)
 AAAAA
```

SPACE$ <(X)>
Devuelve una cadena de X espacios El valor de X debe estar comprendido entre 0 y 255.

```
10 A$ = SPACE$ (5)
20 PRINT A$
```

INSTR <(N, X$, Y$)>
Analiza si la variable Y$ esta contenida en la variable X$, a partir de la posición N
X$ e Y$. pueden tener un maximo de 255 caracteres

Si Y$ no esta contenida en X$, nos dara el valor 0 De o contrario nos dara el numero de la posicion a partir del cual Y$ esta contenido en X$

```
20  A$      ORDENADOR "
20  B$ =  'DENA '
30  PRINT  INSTR  (2,A$,,B$)
     (RUN)
      3
```

LEN <(X$)>
Calcula el numero de caracteres que contiene la cadena X$

```
10 A$ = "ORDENADOR SONY "
20 PRINT LEN (A$)
    (RUN)
     14
10 A = LEN ("SONY ")
20 PRINT A
    (RUN)
     4
```



9. PROCESO DE ERRORES

- **ON ERROR GOTO**
- **ERL/ERR**
- **RESUME**
- **ERROR**

ON ERROR GOTO <nº línea>
Interrumpe el programa principal al producirse un error, pasando a realizar la subrutina de interrupción especificada en <nº línea>

● <nº línea> línea inicial de la subrutina de interrupción

Una vez activada la interrupción todos los errores detectados, incluidos los errores en modo directo (p e Syntax error) irán a la subrutina de error

Para desactivar la interrupción de ERROR debe ejecutarse **ON ERROR GOTO 0.** Los siguientes errores imprimirán en pantalla su mensaje de error correspondiente y detendrán la ejecución del programa

Una instrucción **ON ERROR GOTO 0** que empieza en una subrutina de interrupción de ERROR detendría el programa, imprimiendo el mensaje de error que causó la interrupción.

Si se produce un error durante la ejecución de la subrutina, se imprimirá el mensaje correspondiente y se detendrá la ejecución

Dentro de una subrutina de **ERROR** no puede producirse una interrupción de **ERROR.**

ERL/ERR

Cuando se produce una interrupcion de error, la variable **ERR** contiene el codigo del error producido (ver codigos de error) y la variable **ERL** contiene el numero de linea en el que se origino el error

ERR y **ERL** se utilizan normalmente con la instruccion IF THEN para dirigir el programa hacia una subrutina de error
Si la instruccion que motivo el error fué una instruccion en modo directo, **ERL** sera igual a 65 535 Para comprobar si se ha producido error en una instruccion directa puede utilizarse IF 65535 = ERL THEN... De lo contrario, puede utilizarse:

IF ERL = <numero de linea> THEN...
IF ERR = <código de error> THEN...

Al ser **ERL** y **ERR** variables reservadas, no pueden aparecer a la izquierda del signo igual ( ) en una instruccion **LET**

RESUME
RESUME 0
RESUME NEXT
RESUME <núm linea>
Continua la ejecucion de un programa tras haber rectificado el error indicado

- RESUME o RESUME 0 La ejecución se reanuda en la instrucción que causo el error.

- RESUME NEXT La ejecucion se reanuda en la instruccion siguiente a la que causó el error.

- RESUME <num linea> La ejecucion se reanuda en el numero de linea especificado.

Una instrucción **RESUME** que no este en la subrutina de error, provoca el error: "RESUME without"

ERROR <codigos>
Imprime en pantalla el mensaje de error correspondiente al codigo de error especificado en <codigos>, si este valor es menor de 60 Si <codigo> esta comprendido entre 60 y 255 se utiliza para la definicion de errores por el usuario.

- <codigo> Debe ser un numero entero comprendido entre 1 y 255, siendo 1 al 59, codigos reservados para el **BASIC** y el resto codigos para definicion del usuario

```
10 ON ERROR GOTO 1000

120 IF A$ = "N" THEN ERROR 250

1000 IF ERR   250 THEN PRINT  ESTAS SEGURO?'
```

Si una instruccion de error especifica un código para el cual no se ha definido ningun mensaje de error se producirá el error: "Unprintable error".

La ejecucion de una instruccion de error para la que no hay subrutina de interrupcion causara el mismo error anterior.

```
10 ON ERROR GOTO 500

60 PRINT SQR (—2)

500 IF ERR = 5 THEN RESUME NEXT
```



10. PANTALLA Y GRAFICOS

- SCREEN
- WIDTH
- CLS
- COLOR
- LOCATE
- TAB
- SPC
- POS
- CSRLIN
- KEY ON/OFF
- PSET
- PRESET
- LINE
- CIRCLE
- PAINT
- DRAW
- SPRITE$
- PUT SPRITE
- POINT
- BASE

SCREEN <modo>, <tamano Sprites>, <sonido teclado>, <velocidad de transmisión>, <impresora>
Define los parametros especificados.

● <modo> Selecciona la pantalla para trabajar con texto o con gráficos
● 0 : Modo texto (40 × 24)
● 1 : Modo texto (32 × 24)
● 2 . Modo grafico (256 × 192)
● 3 : Modo grafico (256/4 × 192/4)
Cada punto en modo 3 corresponde a 16 puntos en modo 2 (4 × 4)

● <tamano Sprites> Determina el tamano del Sprite
● 0. 8 × 8 normal
● 1: 8 × 8 ampliado
● 2: 16 × 16 normal
● 3: 16 × 16 ampliado

● <sonido teclado> Habilita o inhibe el sonido producido al pulsar una tecla. ("clik")
● 0. sin sonido
● 1: con sonido

● <velocidad de transmision> Determina la velocidad de transmision de un programa en cassette
● 1:1.200 Baudios.
● 2: 2.400 Baudios.

La velocidad de almacenamiento (cassette), tambien puede determinarse con el comando CSAVE, utilizando la opcion <velocidad>

● <impresoras> Especifica si la impresora conectada es MSX o no
Si la impresora no es MSX, los simbolos graficos son convertidos a espacios.
● 0: MSX
● 1: no MSX

WIDTH <(X)>
Proporciona una pantalla con X caracteres por linea en modo texto. (ver comando **SCREEN**)

● <(X)> numero de caracteres por linea Debe ser un entero comprendido entre 0 y 39 en modo 40 × 24 o entre 0 y 31 en modo 32 × 24.

```
WIDTH 20
```

CLS
Borra el contenido de la pantalla, manteniendo el programa almacenado en memoria.

COLOR <X>, <Y>, <Z>
Define el color de la pantalla.

- <X>: color de los caracteres
- <Y>: color del fondo
- <Z>: color del margen.

El valor de <X>, <Y> y <Z>, debe ser un entero comprendido entre 0 y 15

Si se omite alguno de estos parametros, mantendra el valor anterior

Al inicializar el sistema los valores asignados son X = 15, Y = 4, X = 7. (COLOR 15,4,7)

El color del margen solo puede visualizarse en SCREEN 1, SCREEN 2, y SCREEN 3 (Inicialmente el ordenador esta en SCREEN 0)

Código de colores:

0 — Transparente
1 — Negro
2 — Verde
3 — Verde claro
4 — Azul oscuro
5 — Azul claro
6 — Rojo oscuro
7 — Azul celeste
8 — Rojo
9 — Rojo claro
10 — Amarillo oscuro
11 — Amarillo claro
12 — Verde oscuro
13 — Magenta
14 — Gris
15 — Blanco

LOCATE <X>, <Y>
Situa el cursor en una determinada posicion de la pantalla (solo puede utilizarse en SCRREN 0 y SCREEN 1).

- < X > coordenada horizontal. Debe ser un entero comprendido entre 0 y el valor maximo de la anchura de pantalla menos 1

- < Y > coordenada vertical. Debe ser un entero comprendido entre 0 y 22.

LOCATE se utiliza normalmente para situar el punto inicial de escritura utilizando los comandos **PRINT** e **INPUT** en la siguiente instrucción del programa.

```
10 LOCATE 10 12  PRINT "LOCATE"
10 LOCATE 5 10  INPUT  NOMBRE    A$
```

TAB <(X)>
Desplaza el cursor (X) espacios en sentido horizontal

- <(X)> numero de posiciones a desplazar. Debe ser un entero entre 0 y 255.

TAB solo puede utilizarse con PRINT y LPRINT

```
10 PRINT TAB (10)  BASIC MSX
```

SPC <(X)>
Imprime X espacios en la pantalla

- <(X)> numero de espacios a imprimir. Debe ser un entero entre 0 y 255.

SPC, solo puede utilizarse con PRINT y LPRINT

```
10 PRINT SPC (10)
```

POS <(X)>
Proporciona la coordenada horizontal actual del cursor

- <(X)> Puede ser cualquier caracter no influyendo en el resultado

```
10 PRINT POS(A)
10 IF  POS (1) = 5 GOTO 200
```

CSRLIN
Proporciona la coordenada vertical actual del cursor

```
10  PRINT  CSRLIN
10  IF  CSRLIN = 15 GOTO 200
```

KEY ON/OFF

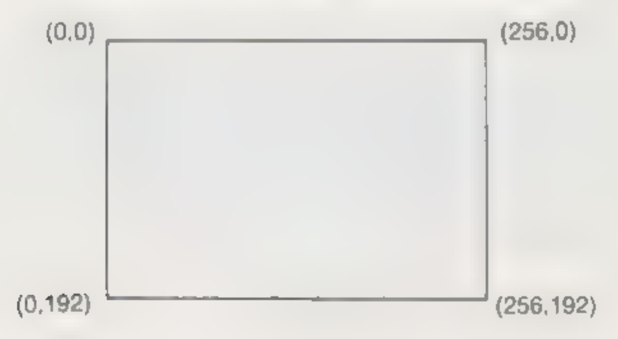
Visualiza o borra el contenido de las teclas de funcion en la ultima linea de la pantalla.

PSET <(coordenadas)>, <color>
Dibuja un punto en las coordenadas especificadas con el color indicado

Solo puede utilizarse en modo grafico (SCREN 2/SCREEN 3)

PRESET <(coordenadas)>, <color>
Borra un punto de la pantalla en las coordenadas especificadas Si se especifica el parametro <color> actua igual que PSET
Solo puede utilizarse en modo grafico (SCREEN 2/SCREEN 3)

NOTA En modo grafico la pantalla queda dividida en una rejilla de 256 puntos horizontales por 192 puntos verticales



```
10 SCREEN 2
20 COLOR 1,1,1
30 FOR A = 1 TO 500
40 X = INT (RND(1)* 255)
50 Y = INT (RND(1)* 191)
60 PSET (X,Y), 15
70 PRESET (X+1, Y+1), 2
80 NEXTA
90 GOTO 30
```

LINE <(coordenada 1)> - <(coordenada 2)>, <color>
Dibuja una linea entre los dos puntos especificados, con el color indicado.
Solo puede utilizarse en modo grafico (SCREN 2/SCREEN 3)

- <(coordenada 1)> coordenadas del punto de inicio
- <(coordenada 2)> coordenadas del punto final
- <color> color de la linea (ver codigo de colores)

Si despues de <color> se especifica el parametro B, dibujara un rectángulo teniendo en cuenta que los puntos dados como coordenadas deben corresponder a una de las diagonales.

Asi mismo si en lugar de especificar el parámetro B se especifica BF, pintara un rectangulo con el color indicado en <color>.

```
10 SCREEN 2
20 LINE (10,10)—(240,10), 1
30 LINE (20,40)—(90,100), 2, B
40 LINE (60,70)—(160,180), 8, BF
50 GOTO 50
```

CIRCLE <(coordenadas)>, <radio>, <color>, <angulo de inicio>, <ángulo final>, <excentricidad>

Dibuja circulos elipses y arcos segun los parametros especificados Los unicos parametros obligatorios son <coordenadas> y <radio> (Solo puede utilizarse en modo grafico SCREEN 2/SCREEN 3)

- <(coordenadas)> coordenadas del punto central
- <radio>: radio
- <color>: color (ver código de colores)
- <angulo inicial> <angulo final> angulo inicial y final de la figura

Deben expresarse en radianes, entre 0 y 2 π Si el angulo es negativo, la elipse se conectara al punto central con una linea, y el angulo sera tratado como si fuera positivo (Observe que esto es distinto a sumar 2π).

<excentricidad> Relación entre los ejes horizontal y vertical Si <excentricidad> 2, el eje horizontal será la mitad del vertical, permitiendo así definir la forma deseada en la elipse.

```
 5 SCREEN 2
10 CIRCLE (125, 96), 70
10 CIRCLE (125, 96), 70, 1, , , 1.2
10 CIRCLE (125, 96), 70, 1, 0, 2*3.14, 2
10 CIRCLE (125, 96), 70, 1, 0, 2*3.14, 5
10 CIRCLE (125, 96) 70, 1, , ,0
```

NOTA Para dibujar un circulo debe especificarse en excentricidad la relación 1.2.

PAINT <coordenadas>, <color 1>, <color 2>
Rellena de color un contorno cerrado
Sólo puede utilizarse en modo grafico (SCREEN 2/SCREEN 3)

- <coordenadas> coordenadas del punto desde donde se empezará a pintar
  F corresponden a un punto interior del contorno cerrado rellenará de color el interior de dicho contorno, dé lo contrario pintará el exterior

- <color 1> color con el que se rellenara el contorno (interior o exterior).

● <color 2> color que limita la superficie a pintar con <color 1>
En alta resolucion este parametro no tiene efecto debiendo ser el color del contorno a pintar del mismo color que <color 1>.

Si no se especifican los parametros de color el color utilizado sera el primer parametro del comando COLOR actual (color de los caracteres).

```
10 SCREEN 3
20 CIRCLE (125, 96), 50, 1
30 PAINT (125, 96), 2, 1
40 PAINT (10, 10), 10, 1
50 GOTO 50
```

```
10 SCREEN 2
20 CIRCLE (125, 96), 50, 1
30 PAINT (125, 96), 1
40 GOTO 40
```

DRAW
Sentencia de ejecucion de los comandos del Macro lenguaje grafico (ver Anexo de graficos).

SPRITE$
Variable especial del sistema utilizada para la definicion de graficos SPRITE. (Ver Anexo de gráficos)

PUT SPRITE <nº plano> <coordenadas> <color>
Asigna a un SPRITE los atributos especificados
(Ver Anexo de graficos)

POINT <(x,y)>
Ofrece el codigo de color de un punto de la pantalla

● <(x,y)>: coordenadas del punto
S estan fuera del rango de visualizacion el valor obtenido es — 1

BASE <(x)>
Ofrece la primera direccion de las tablas del procesador de video (VDP)

● <(x)> Debe ser un numero entero positivo comprendido entre 0 y 19

Para mas informacion ver anexo 3 (Informacion Tecnica)

11. SONIDO

- **BEEP**
- **SOUND**
- **PLAY**
- **PLAY (N)**

BEEP
Genera el sonido «Bip». (CTRL-G).

```
10 FOR I   1 TO 10
20 BEEP
30 FOR X   1 TO 100  NEXT X
40 NEXT I
```

SOUND <registro del PSG>, <valor>
Escribe directamente el valor especificado en el registro indicado del Procesador generador de sonido. (PSG).

Registros del PSG		
Nº de registro	Funcion	Valor
0	Frecuencia canal A.	0-255
1		0-15
2	Frecuencia canal B	0-15
3		0-15
4	Frecuencia canal C	0-255
5		0-15
6	Frecuencia de ruido	0-31
7	Selecciona un canal para generación de tonos y ruido	0-63
8	Volumen canal A	0-15 La variacion de volumen tendrá lugar cuando se seleccione 16
9	Volumen canal B	
10	Volumen canal C	
11	Frecuencia del patrón de variacion de volumen	0-255
12		0-255
13	Selección del patrón de variacion de volumen	0-14

```
 10 FOR I = 1 TO 50
 20 SOUND 6, 0
 30 SOUND 7, 55
 40 SOUND 8, 16
 50 SOUND 11, 23
 60 SOUND 12, 2
 70 SOUND 13, 9
 80 FOR X = 1 TO 100  NEXTX
 90 NEXT I
100 FOR I = 1 TO 10
110 SOUND 7, 55
120 SOUND 6, 0
130 SOUND 8, 16
140 SOUND 11, 50
150 SOUND 12, 47
160 SOUND 13, 0
170 FOR X = 1 TO 100. NEXTX
180 NEXT I
190 GOTO 10
```

PLAY <expresión para VOZ 1>, <expresión para VOZ 2>, <expresión para VOZ 3>

Es la sentencia de ejecución de los comandos del macrolenguaje musical. Puede interpretar tres voces simultáneamente, en un rango de ocho octavas. (ver anexo II: sonido).

PLAY (N)

Comprueba si se está ejecutando música.

- Si N = 1, 2 o 3, da el valor -1 si se está ejecutando música; de lo contrario da el valor 0

- Si N = 0 el estado de cada comando del macrolenguaje musical (-1 ó 0) se opera con OR exclusive y da el resultado obtenido



12. ALMACENAMIENTO EN CASSETTE

- **CSAVE**
- **CLOAD**
- **CLOAD?**
- **SAVE**
- **LOAD**
- **BSAVE**
- **BLOAD**
- **MERGE**
- **MOTOR ON/OFF**

CSAVE "< Nombre del Programa >", < velocidad >
Almacena en cassette un programa **BASIC** con el nombre especificado en "<nombre del Programa>"

El formato utilizado es el Binario comprimido que ocupa menos espacio que el ASCII, aunque algunos tipos de acceso requieren que los ficheros esten almacenados con este formato. Por ejemplo si queremos mezclar dos programas, uno de ellos debe estar en código ASCII. (ver comando MERGE).
El comando SAVE realiza el almacenamiento en formato ASCII

- "< nombre del programa >" Nombre asignado al programa en el momento de grabarlo en cassette. Para cargar el programa desde cas-

sette a memoria, utilizaremos este mismo nombre El numero maximo de caracteres significativos es 6.

- <velocidad> Determina la velocidad de grabacion
  1: 1.200 Baudios
  2 2 400 Baudios
  Este parametro es optativo ya que si no se especifica, el ordenador asignará por defecto el valor 1

```
CSAVE "PROG"
CSAVE "PROG1", 2
```

CLOAD "<nombre del programa>"

Carga el programa especificado en «<nombre del programa>», desde cassette a la memoria interna del ordenador, borrando el contenido actual de la memoria Anteriormente dicho programa fue almacenado en cinta con el comando CSAVE

```
CLOAD "PROG"
```

CLOAD? "<nombre del programa>"
Carga un programa almacenado en memoria externa (cassette) a la memoria interna del ordenador sin borrar el contenido actual de la memoria interna estableciendo una comparacion entre el programa que esta cargando y el que esta residene en memoria

Su funcion es comprobar si la grabacion efectuada con CSAVE ha sido correcta En caso de grabacion incorrecta se producira el error "Verify error."

Una vez grabado el programa en cassette mediante CSAVE, se rebobina la cinta hasta el punto inicial de grabacion y se teclea en el ordenador **CLOAD**? seguido del nombre del programa (enmarcado entre comillas) Despues, daremos la orden de ejecucion pulsando la tecla **RETURN** y pondremos el cassette en marcha
Si la grabacion ha sido correcta aparecera «OK» en caso contrario se imprimira el mensaje de error correspondiente

```
CLOAD? "PROG"
OK
```

SAVE "<nombre del programa>"

Almacena en cassette un programa BASIC en formato ASCII con el nombre especificado en '<nombre del programa>'

```
SAVE "PROG"
```

LOAD "<nombre del programa>"
Carga el programa, especificado en "<nombre del programa>', desde cassette a la memoria interna del ordenador. Anteriormente, dicho programa fue grabado con **SAVE.**

```
LOAD "PROG"
```

Si después de "<nombre del programa>" se añade , R, el programa se ejecutará automáticamente sin necesidad de utilizar el comando **RUN.**

```
LOAD "PROG 1", R
```

BSAVE <CAS Nombre del programa> <dirección inicial> <dirección final>, <dirección de ejecución>.

Almacena en cassette un programa en lenguaje máquina localizado en las posiciones de memoria indicadas (Desde <dirección inicial> hasta <dirección final>).

Si se omite el parámetro <dirección de ejecución> se asignará por defecto el valor de <dirección inicial> sino, el programa se ejecutará a partir de esta posición cuando sea cargado con **BLOAD.**

```
BSAVE "CAS TEST", &HA000, &HAFFF
BSABE "CAS GAME" &HE000, &HE0FF, &HE020
```

BLOAD "<CAS. Nombre del programa>"
Carga desde cassette un programa en lenguaje maquina, que fue grabado mediante **BSAVE.**

Si despues de "CAS: Nombre del programa" se anade ", R" el programa se ejecutara inmediatamente desde la direccion que se especifica al grabarlo con **BSAVE.** El programa, se almacenara en las mismas posiciones de memoria especificados en **BSAVE.**

```
BLOAD "CAS:TEST", R
```

MERGE "<nombre del programa>"
Mezcla el programa especificado en <nombre del programa> con el residente en memoria. El programa especificado debe ser un programa en formato ASCII (almacenado mediante SAVE).

Si ambos programas tienen algun numero de linea comun, el contenido final de esta linea sera el correspondiente al programa en formato ASCII.

```
MERGE "VIDEO"
```

MOTOR ON/OFF

Pone en marcha o detiene el cassette cuando se utiliza el control remoto.

Si **MOTOR** no va acompanado de argumento (**ON/OFF**) pondra en marcha/detendra el cassette secuencialmente. Es decir, si el cassette esta parado lo pondra en marcha y viceversa.

Si se utiliza el control remoto, todas las instrucciones de ficheros como **CLOAD, CSAVE, PRINT#, INPUT#**, etc. activaran automaticamente el cassette.

CONSEJOS PRACTICOS PARA LA UTILIZACION DEL CASSETTE

- Utilice siempre un cassette monofonico. En caso de utilizar un cassette stereo perderia parte de la informacion.

- Ponga el volumen del cassette a un nivel MEDIO ALTO. La grabacion estara sometida a interferencias si el volumen esta demasiado bajo. Por otra parte, si el volumen esta demasiado alto, puede producirse distorsion en la señal de salida.

- Si su cassette tiene control de TONO, situelo en una posicion alta (HIGH), ya que la grabacion contiene principalmente frecuencias altas.

- Compruebe que las pilas de su cassette estan en buen estado. Recomendamos la utilizacion de un alimentador.

- Compruebe que el cabezal esta en buenas condiciones de limpieza.

- Procure utilizar siempre el mismo cassette para la grabacion y carga de programas, ya que entre distintos aparatos pueden existir pequeñas diferencias (velocidad del motor, ajuste de cabezales, etc.), que pueden ocasionar errores de carga.



13. FICHEROS

- **OPEN #**
- **MAXFILES**
- **PRINT #**
- **PRINT # USING**
- **INPUT #**
- **INPUT$(n, #)**
- **LINE INPUT #**
- **EOF**
- **CLOSE #**

OPEN "<periferico> <nombre fichero>" FOR <modo> AS # <num fichero>

Abre un fichero en el periférico especificado, asignandole un numero determinado que será utilizado por otros comandos de Entrada/Salida para especificar el fichero con el que van a trabajar

- <periférico>: CAS: cassette
  CRT: pantalla.
  GRP: pantalla de gráficos
  LPT: impresora

- <modo> OUTPUT salida secuencial de datos
  INPUT: entrada secuencial de datos
  APPEND: adición de datos.

● <num fichero> Debe ser un numero entero comprendido entre 1 y el valor especificado en la sentencia MAXFILES. Este numero quedara asociado al fichero en cuestion pudiendo hacer referencia a el otros comandos de E/S (PRINT #, INPUT #...) utilizando dicha asignacion.

```
OPEN "CAS:DAT" FOR OUTPUT AS # 1
```

Abre el fichero DAT almacenado en cassette (CAS) en modo salida (OUTPUT), asignandole el número 1. (# 1)

NOTA: Para introducir texto trabajando con pantalla de grafico (SCREEN 2/SCREEN 3) debera realizar un OPEN «GRP:» FOR OUTPUT AS # 1.

```
10 OPEN "GRP:" FOR OUTPUT AS # 1
20 SCREEN 2
30 PRESET (50,100)
40 PRINT # 1,"BASIC"
50 GOTO 50
```

La Linea 30 situa el punto inicial de escritura.

MAXFILES = <num. ficheros>
Especifica el numero maximo de ficheros que estaran abiertos simultáneamente en un programa.

Si MAXFILES = 0 solo podrán utilizarse los comandos SAVE y LOAD.

El valor asignado por defecto es 1 (MAXFILES = 1).

MAXFILES = 5 indica que en el transcurso del programa podemos tener un maximo de 5 ficheros abiertos (OPEN) simultaneamente.

PRINT # <num. fichero>, <lista de datos>
PRINT # <num. fichero>, **USING** <expresion>; <lista de datos>

Escribe datos en el fichero especificado que posteriormente podrán ser leidos mediante la sentencia INPUT #.

Antes de efectuar una operacion de escritura debe abrirse el fichero en modo salida (OUTPUT).

```
10 OPEN  CAS DAT  FOR OUTPUT AS # 1
20 INPUT # 1 PEDRO 23
```

(Ver PRINT/PRINT USING para mas detalles)

INPUT # < num fichero > < lista de variables >

Lee datos desde el fichero especificado asignandolos a las variables indicadas en <lista de variables>.
Los datos del fichero estaran almacenados en el mismo orden en que fueron escritos por el comando **PRINT** # por lo que para asignarlos a las variables del programa deberemos conocer dicho orden

- < num fichero > Es el numero con que fue abierto (OPEN) el fichero.

- < lista de variables > Pueden ser numericas o alfanumericas, debiendo estar separadas por coma.

Antes de efectuar una operacion de lectura, debe abrirse el fichero en modo entrada. (INPUT):

```
10 OPEN  CAS DAT  FOR INPUT AS # 1
20 INPUT # 1 A$ B$ C D$, E
```

INPUT$ (n, # <num. fichero>)

Lee una secuencia de *n* caracteres en el fichero especificado asignandolos a una variable.

- < numero de fichero > es el numero con el que fue abierto (OPEN) el fichero

Antes de efectuar una operacion de lectura, debe abrirse el fichero en modo entrada (INPUT):

```
10 OPEN "CAS DAT  FOR INPUT AS # 1
20 A$  INPUT$ (5 # 1)
```

LINE INPUT # <num. fichero>, <variable>
Lee una línea entera (hasta 254 caracteres) del fichero especificado sin utilizar delimitadores asignando la cadena a una variable alfanumérica

- <num fichero> es el número con el que fue abierto (OPEN) el fichero.

- <variable> es la variable a la que se asignará la secuencia de caracteres leídos. Debe ser una variable alfanumérica

LINE INPUT # lee todos los caracteres hasta encontrar un **RETURN.** El próximo LINE INPUT # empieza a leer desde este punto hasta el próximo **RETURN** y así sucesivamente

Este comando es especialmente útil en aquellos programas que deben colocar cada línea de un fichero en un campo determinado o para programas que deben leer datos de otro programa almacenado en ASCII

Antes de efectuar una operación de lectura debe abrirse el fichero en modo entrada **(INPUT)**.

```
10 OPEN "CAS DAT "FOR INPUT AS # 1
20 LINE INPUT # 1 A$
```

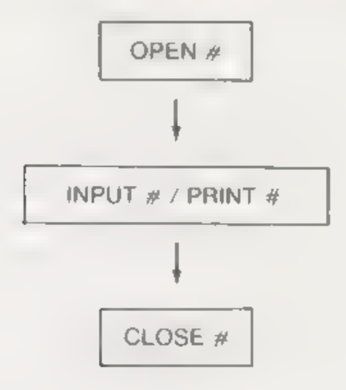
EOF (<núm. fichero>)
EOF tomará el valor —1 si se ha llegado al final de un fichero secuencial (EOF = -1).
Utilice EOF durante la lectura de un fichero para evitar errores tipo "Input past end".

```
IF EOF (1)   -1 THEN CLOSE # 1
```

CLOSE # <num. fichero>, <num fichero>
Cierra el fichero o ficheros especificados en <num fichero>
Si no se indica <num fichero> cerrará todos los ficheros abiertos

```
CLOSE # 1    cierra el fichero 1
CLOSE        cierra todos los ficheros abiertos
```

NOTA: Las operaciones a realizar cuando trabajamos con ficheros pueden esquematizarse segun el siguiente organigrama





ANEXOS:

1. GRAFICOS
2. SONIDO
3. INFORMACION TECNICA
4. PROGRAMAS
5. CODIGOS DE ERROR



ANEXO 1: GRAFICOS

INTRODUCCION

Además de las funciones gráficas vistas anteriormente (LINE, CIRCLE, PSET), el HIT BIT nos ofrece nuevas opciones

— Macro lenguaje para gráficos
— Gráficos SPRITE

Todas las funciones gráficas pueden utilizarse conjuntamente trabajando en modo gráfico (SCREEN 2, SCREEN 3)

1. MACRO LENGUAJE PARA GRAFICOS

Utilizando este macro lenguaje podremos acceder a todos los puntos de la pantalla en modo gráfico (256 × 192) para realizar cualquier dibujo Para ello utilizaremos la sentencia DRAW que tiene el siguiente formato

DRAW <expresión string>

Los comandos del macro lenguaje son letras que indican el movimiento a realizar, y forman parte de la expresión string El string define un objeto que será dibujado al ejecutar la sentencia DRAW

Al utilizar los comandos de movimiento (ver tabla I), hay que tener en cuenta que el movimiento se iniciará desde el último punto referenciado Es decir si inicialmente estamos en (5 20) y ejecutamos un R5 (desplazamiento de 5 puntos a la derecha) el punto final será (10 20), siendo este el punto de inicio del próximo comando

TABLA I: Comandos de movimiento

n, indica la distancia de movimiento. El numero de puntos desplazados sera **n** veces el factor de escala (indicando por el comando S)

Mx y, indica desplazamiento absoluto o relativo. Si se indica X + o X — el desplazamiento sera relativo, de lo contrario el desplazamiento sera absoluto.

La proporcion de la pantalla es 1:1. Es decir que 8 puntos horizontales tienen la misma longitud que 8 verticales.

COMANDO	MOVIMIENTO
U n	↑
D n	↓
L n	←
R n	→
E n	↗
F n	↘
G n	↙
H n	↖
My y	**X,Y**

Todos los comandos indicados al ser ejecutados mediante la sentencia DRAW dibujaran siempre el recorrido programado. Si en algun momento queremos desplazarnos hasta un punto determinado sin dibujar la trayectoria seguida o queremos dibujarla volviendo al punto inicial utilizaremos los siguientes prefijos.

B desplazamiento sin dibujar trayectoria
N desplazamiento dibujando trayectoria y volviendo al punto de inicio.

Asi por ejemplo, normalmente lo primero que tendremos que hacer para empezar un dibujo sera situar el cursor en la posicion inicial sin dibujar la trayectoria seguida. Esto podemos hacerlo mediante el comando BMx, y, donde «x y» son las coordenadas iniciales hasta donde queremos trasladarnos (Mx y dibujaria la trayectoria seguida hasta el punto x,y).

Otros comandos que pueden utilizarse con este macro lenguaje son

An: Permite movimiento en el angulo n, que puede ser de 0 a 3 (0 = 0°, 1 = 90°, 2 = 180°, 3 = 270°).



Cn : Da color a os puntos dibujados (n 0 — 15) Ver código de colores

Sn : Indica el factor de escala El valor de n debe ser un numero entero comprendido entre 0 y 255 El factor de escala sera n/4 Por ejemplo, si n = 1, el factor de escala será 1/4.
El factor de escala multiplicado por la distancia indicada (n) en los comandos de movimiento U D,L R,E F G H,M nos dara el numero de puntos desplazados
Si no se indica factor de escala el valor por defecto será n 4 (S4)

X < variable string > Este comando permite ejecutar un segundo string desde otro string

Dibujo de un cuadrado·

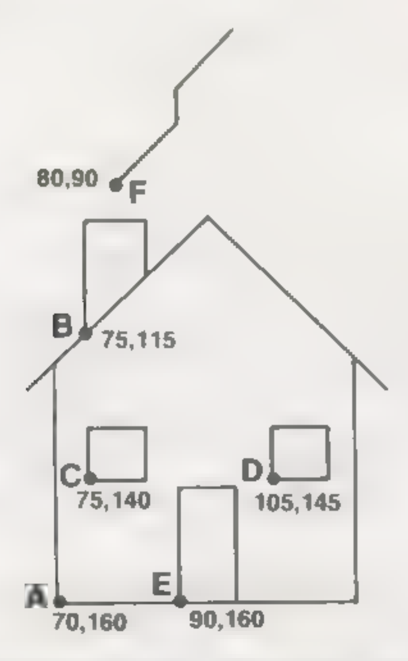
```
10 SCREEN 2
20 A$      U80R80D80L80
30 DRAW  BM70  150XA$.
40 GOTO 40
```

BM 70 150 coloca el punto inicial del dibujo en (70, 150) de otra forma, el punto inicial seria (0, 0) y no podriamos visualizarlo

El comando X es una parte muy util de **DRAW**, pues permite definir partes de un objeto separadas del conjunto Asi por ejemplo si en un dibujo hay un elemento que debe aparecer en distintas posiciones podemos definir este elemento con el comando X evitando asi el tener que programarlo cada vez que queremos dibujarlo

Dibujo de una casa

1°) Hacemos el dibujo en una hoja de papel cuadriculada con un factor de escala de 1 cuadro 5 puntos y fijamos el punto de inicio en (A)



2º) Hacemos el programa del dibujo dividiendolo para mayor comodidad en 5 partes:

A$ = contorno casa
B$ = chimenea
C$ = ventana
D$ = puerta
E$ = humo

```
A$ = "U40NGH5E25F30H5D40L50"
B$ = "U20R10D10"
C$ = "U10R10D10L10"
D$ = "U20R10D20"
E$ = "E10U5E10"
```

3°) Determinamos los puntos de inicio de cada objeto

A$ — CASA ———————— A (70, 160)
B$ — CHIMENEA ———————— B (75, 115)
C$ — VENTANA 1 ———————— C (75, 140)
C$ — VENTANA 2 ———————— D (105, 145)
D$ — PUERTA ———————— E (90, 160)
E$ — HUMO ———————— F (80, 90)

Antes de hacer el dibujo de cada objeto colocaremos el cursor en los puntos correspondientes mediante el comando **MBx,y.**

El programa completo será el siguiente.

```
 10 SCREEN 2
 20 A$ = "U40NG5E25F30H5D40L50"
 30 B$ = "U20R10D10"
 40 C$ = "U10R10D10L10"
 50 D$ = "U20R10D20"
 60 E$ = "E10U5E10"
 70 DRAW "BM70, 160XA$;"
 80 DRAW "BM75, 115XB$;"
 90 DRAW "BM75, 140XC$;"
100 DRAW "BM105, 145XC$;"
110 DRAW "BM90, 160XD$;"
120 DRAW "BM80, 90XE$;"
130 GOTO 130
```

Para mayor comodidad podemos hacer todo el dibujo con una sola instruccion. Para ello sustituiriamos las lineas 70-130 por

```
70 DRAW "BM70, 160XA$; MB 75, 115 XB$; BM 75, 140XC$; BM 105,
145 X C$; BM 90, 160 XD$; BM 80, 90 XE$;"
80 GOTO 80
```

PROGRAMA EJEMPLO UTILIZACION MACRO LENGUAJE PARA GRAFICOS

```
5  REM $MACRO LENGUAJE GRAFICO$

10  SCREEN2

20  REM A$ = PERFIL CASA.

30  A$ =  BM70 130U60BD60R60U60BL30BU40NF40NG40D2NF38NG38"

40  REM B$ = PUERTAS/VENTANAS

50  B$ =  BM80 70R10D20L10U20BR30R10D20L10U20BM90,
100R20D30BL20U30BM82 70D20BR2U20BR2D20BR2U20BM112,
70D20BR2U20BR2D20BR2U20BM92,100D30BR2U30BR2D30BR2U30BR2
D30BR2U30BR2D30BR2U30BR2D30"

55  REM C$ = CHIMENEA/HUMO.

58 C$ = 'BM8050U15R10D5BM90 30E10U10E10BD10R10D10
L10U10DB10BR20U10F10U10BR10F5NE5D5

60  REM D$ = PAJAROS.

70  D$ = "BM170 30F10E10BM190,40F10E10"

80  REM E$ = CARTEL. HIT BIT 55

90  E$   ' BM170 120R60D50L60U50BM180,
140  U10D5R5D5BR5U5BU2U1BM195,
140  U10NL2NR2BR10D10U5R5D5L5BR10U5BU2U1BM220,
140  U10NL2NR2BM190 160R5U5L5U5R5BD10BR10R5U5L5U5R5BM190,
130  D20R20U20"

100  REM F$ = CAMINO.

110  F$   "BM90,140F10G10F10BM110,140F10G10F10 '

150  DRAW"XA$;"

160  DRAW"XB$,"

170  DRAW' XC$.
```

```
180 DRAW"XD$,"
190 DRAW"XE$;"
200 DRAW"XF$:"
250 FOR I =0 TO 5000
260 NEXT I
270 A = RND(1)*15
280 B = RND(1)*15:C = RND(1)*15
290 COLOR A,B,C
300 GOTO 10
```

2. GRAFICOS SPRITE

Hasta ahora, para representar gráficos en movimiento hemos utilizado la sentencia **PRINT.** Esta forma de representacion para la animación gráfica, resulta muy limitada debido principalmente a que al utilizar la sentencia **PRINT,** los objetos representados deben crearse a partir de los simbolos gráficos existentes, por lo que la forma y resolucion del objeto no puede ser lo buena que se requiere
Por otra parte, el movimiento del grafico por la pantalla requiere un numero de instrucciones para controlar su situación

Veamos un ejemplo el siguiente programa simula el vuelo de una nave espacial.

```
10 CLS
20 FOR A = 0 TO 26
30 LOCATE A,10
40 PRINT "XOX"
50 FOR I = 1 TO 40
60 NEXT I
70 LOCATE A, 10
80 PRINT "
90 NEXT A
```

Al ejecutar el programa en el ordenador se observar que la nave espacial (XOX) no ha salido muy favorecida, debido a las causas citadas anteriormente y el control de su movimiento puede llegar a complicar bastante el programa si pretendemos que el objeto se mueva por distintas direcciones. Todos estos inconvenientes quedan subsanados utilizando los gráficos **SPRITE** del **HIT BIT,** ya que podemos dar al objeto la forma deseada situarlo en cualquier posición de la pantalla, darle una dirección y velocidad,... etc.

¿Qué es un SPRITE?
Un SPRITE, es un objeto programable, que puede definirse fácilmente en **BASIC.**

SPRITE, es el nombre que recibe la forma de lograr imágenes más naturales y más fácilmente en un ordenador. En los gráficos convencionales las imágenes se componen en una pantalla simple, como si se dibujara en una hoja de papel. Con los SPRITES el ordenador tiene varios planos cada uno de los cuales contiene su propia imagen. En el caso del **HIT BIT,** disponemos de 32 planos distintos.

La manera más sencilla de representar estos planos consiste en imaginarse láminas de plástico transparente. Si la lámina más cercana al observador contiene el dibujo de una estrella, mientras que la última incluye la imagen de una nave espacial, ésta será vista pasar por detrás de la estrella, como si realmente estuviera flotando en el espacio.

Al poner varios elementos en planos separados, podrán crearse efectos tridimensionales muy convincentes.

Otra característica importante de este tipo de gráficos es que para darles movimiento tan sólo hay que especificar una velocidad y una dirección, olvidándonos por completo de cómo fue definido.

Veamos cómo mejora el aspecto de nuestra nave espacial utilizando SPRITES. Ejecute el siguiente programa.

```
10 COLOR 15, 4, 7
20 SCREEN 2,0
30 CLS
40 SPRITE $ (0)    CHR$(&B00000000) +
+ CHR$(&B00000000) + CHR$(&B00100100) +
+ CHR$(&B00111100) + CHR$(&B01011010) +
```

```
+ CHR$(&B10011001) + CHR$(&B01111110) +
+ CHR$(&B00100100)
50 PUT SPRITE 0, (5 70)
60 FOR I   1 TO 256
70 PUT SPRITE 0 STEP (1 0) 15
80 NEXT I
```

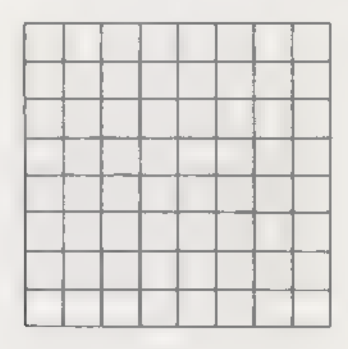
Comandos para la utilización de SPRITES en el HIT BIT

Los comandos utilizados en el **HIT BIT** para la realizacion de graficos SPRITE, son los siguientes

- SPRITE $ (n)
- PUT SPRITE
- ON SPRITE GOSUB
- SPRITE ON/OFF/STOP

SPRITE $ (n) – <definición SPRITE>
Define la forma del objeto El numero de plano (n) debe ser un numero entero entre 0 y 31.

Para definir el SPRITE nos auxiliaremos de una cuadricula de 8 × 8



Dentro de esta cuadricula dibujaremos el SPRITE y a continuacion al lado de cada linea codificaremos su contenido segun el siguiente criterio:

— Si una casilla esta en blanco le asignaremos el valor 0
— Si una casilla esta pintada le asignaremos el valor 1

Finalmente como estos valores obtenidos son binarios se lo indicaremos al ordenador mediante el prefijo &B (si conoce el lenguaje binario puede codificar estos valores en decimal). La definicion del **SPRITE,** estara formada por la suma de los 8 CHR$ correspondientes a los valores decodificados.

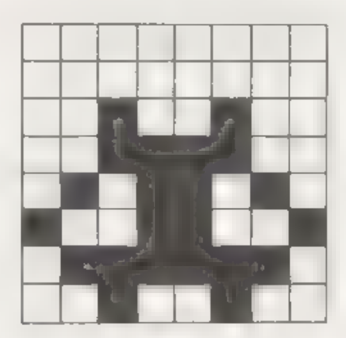
Veamos como construiriamos la nave espacial del ejemplo anterior

1º Hacemos el dibujo en la cuadricula

2º Decodificamos las lineas de la cuadricula

3º Definimos el SPRITE como la suma de los 8 CHR$ decodificados

1º Dibujo



2º Decodificacion

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	1	0	0	1	0	0
0	0	1	1	1	1	0	0
0	1	0	1	1	0	1	0
1	0	0	1	1	0	0	1
0	1	1	1	1	1	1	0
0	0	1	0	0	1	0	0

3º **Asignacion**

SPRITE$ (0) = CHR$ (&B00000000) +
+ CHR$ (&B00000000) + CHR$ (&B00100100) +
+ CHR$ (&B00111100) + CHR$ (&B01011010) +
+ CHR$ (&B10011001) + CHR$ (&B01111110) +
+ CHR$ (&B00100100)

Una forma mas comoda de definir el SPRITE es utilizando *READ/DATA*

```
 10 COLOR 15,4,7
 20 SCREEN 2,0
 30 CLS
 40 FOR X = 1 TO 8
 50 READ A$(X)
 60 B$ – B$ +CHR$ (VAL("&B" + A$(X))
 70 NEXT X
 80 SPRITE$(0) = B$
 90 PUT SPRITE 0, (5,70)
100 FOR I – 1 TO 256
110 PUT SPRITE 0, STEP (1,0), 15
120 NEXT I
130 DATA 00000000
140 DATA 00000000
150 DATA 00100100
160 DATA 00111100
170 DATA 01011010
180 DATA 10011001
190 DATA 01111110
200 DATA 00100100
```

Si queremos que nuestra «nave espacial» aparezca mas grande, sustituiremos la línea 20 por

```
20 SCREEN 2 1 (ver sentencia SCREEN)
```

Tambien podemos trabajar con **SPRITES** de 16 × 16 (ampliables con **SCREEN** 2 3) El procedimiento sera el mismo que en 8 × 8 teniendo en cuenta que la cuadricula sera de 16 × 16 y por lo tanto tendremos 16 lineas de codificacion de 16 bits cada una El principio del programa, debemos indicar si trabajamos con **SPRITES** de 8 × 8 o 16 × 16 y si van a ser ampliables.

```
SCREEN X, 0 —  8 × 8  normal
SCREEN X, 1 —  8 × 8  ampliado
SCREEN X, 2 — 16 × 16 normal
SCREEN X, 3 — 16 × 16 ampliado
```

PUT SPRITE

Con esta sentencia definimos los atributos del SPRITE (situacion, movimiento, color...). Su estructura es la siguiente.

```
PUT SPRITE N, (X, Y), <Color>, <nº SPRITE>
```

N = nº plano (0 - 31)
(X, Y) = situacion del SPRITE
Color = color del SPRITE
nº SPRITE indica el SPRITE al que vamos a asignar los atributos anteriores Este numero debe ser menor a 256 si el tamano del SPRITE es 0 o 1 y menor a 64 si el tamano es 2 o 3 (ver sentencia **SCREEN**)

Para poner el **SPRITE** en movimiento le indicaremos los saltos (**STEPS**) que debe realizar a partir de su posicion inicial As pues normalmente utilizaremos 2 veces la sentencia PUT SPRITE una para colocarlo en la posicion inicial y otra para indicarle el movimiento a realizar

En el ejemplo II la linea 50 coloca el **SPRITE** en la posicion inicial (5 70) y en la linea 70 le especificamos que desde la posicion anterior debe moverse con saltos de (1 0) (desplazamiento horizontal) Si en lugar de STEP (1 0) especificamos STEP (0 1), el movimiento sera vertica

Si queremos que nuestra nave espacial se mueva siguiendo la trayectoria de una sinusoide, cambiaremos las siguientes lineas de programa del ejemplo anterior.

```
60 FOR X = 11 TO 256 STEP 3
70 PUT SPRITE 0, (X, SIN (X/4)*20 + 70)
```

Con las siguientes modificaciones obtendremos un movimiento aleatorio

```
50 PUT  SPRITE 0, (127,96)
60 X    INT (RND)(1)*20)*(-1)    Y Y    INT(RND(1)*20)*(-1)·X
70 PUT  SPRITE 0, STEP (X,Y)
80 FOR  T = 1 TO 50  NEXT
90 GOTO  60
```

ON SPRITE GOSUB <nº linea>
Esta sentencia hara que el programa principal salte a la subrutina de servicio, cuya linea de inicio es la indicada en <nº de linea>, cuando «chocan» dos **SPRITES** si anteriormente se ha habilitado la interrupción mediante SPRITE ON

SPRITE ON/OFF/STOP

Para activar/desactivar las interrupciones de **SPRITES.**

SPRITE ON: Activa la interrupción
SPRITE OFF: Desactiva la interrupción
SPRITE STOP Si se produce interrupcion, la memoriza, pero no salta a la subrutina de servicio hasta que se active la interrupción mediante **SPRITE ON**

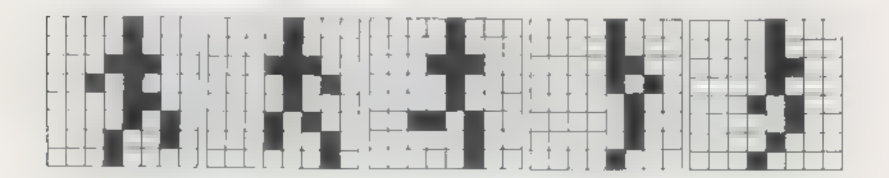
Antes de la sentencia **ON SPRITE GOSUB,** hay que especificar si la interrupcion esta activa o no **(SPRITE ON/OFF/STOP).** Si se produce interrupcion, con **SPRITE ON,** automáticamente se producira un **SPRITE STOP,** y pasaremos a realiar la subrutina de servicio

VARIACION DE LA FORMA DEL OBJETO CON EL MOVIMIENTO

Hasta aqui, hemos visto cómo mover un **SPRITE** por la pantalla sin variar su forma inicial Si queremos que nuestro objeto de más sensacion de

movimiento (moviendo las piernas y los brazos al andar, por ejemplo), tendremos que crear tantos **SPRITES** como movimientos distintos tengamos. Es decir, tendremos que hacer un estudio del movimiento a realizar y definir un **SPRITE** para cada movimiento (imagínese el movimiento en cámara lenta).

Veamos como visualizar un **SPRITE** que está «corriendo».
Los movimientos realizados al correr serán 5.



Por lo tanto utilizaremos 5 SPRITES (SPRITES (0) SPRITES (4)).
El programa será el siguiente

```
10 COLOR 15,4,7
20 SCREEN 1,0
30 CLS
40 FOR L=0 TO 4
50 FOR I=1 TO 8
60 READ D$(L)
70 S$(L)=S$(L)+CHR$(VAL("&B"+D$(L)))
80 NEXT I
90 SPRITE$(L)=S$(L)
100 NEXT L
100 PUT SPRITE 0,(120,90),15,0
120 FOR I = 0 TO L-1
130 PUT SPRITE 0, STEP(2,0),15,I
140 FOR T = 1 TO 20:NEXT
150 NEXT I:GOTO 120
160 DATA 00001000
170 DATA 00001000
180 DATA 00011100
190 DATA 00101000
200 DATA 00001100
210 DATA 00001010
220 DATA 00010010
230 DATA 00010000
```

```
240 REM
250 REM
260 DATA 00001000
270 DATA 00001000
280 DATA 00011100
290 DATA 00001010
300 DATA 00001000
310 DATA 00010100
320 DATA 00010010
330 DATA 00000010
340 REM
350 REM
360 DATA 00001000
370 DATA 00001000
380 DATA 00011110
390 DATA 00001000
400 DATA 00001000
410 DATA 00110100
420 DATA 00000100
430 DATA 00000100
440 REM
450 REM
460 DATA 00001000
470 DATA 00001000
480 DATA 00001100
490 DATA 00001010
500 DATA 00000100
510 DATA 00000100
520 DATA 00000100
530 DATA 00001000
540 REM
550 REM
560 DATA 00001000
570 DATA 00001000
580 DATA 00001100
590 DATA 00001000
600 DATA 00010100
610 DATA 00000100
620 DATA 00001000
630 DATA 00010000
```



ANEXO 2: SONIDO

INTRODUCCION

Una de las características principales del HIT BIT es que interna mente, lleva incorporado un Procesador generador de sonido que le permitirá crear fácilmente sus propias melodías utilizando un Macro lenguaje Musical, muy similar al visto anteriormente para dibujar Para ello dispondremos de 8 octavas, 3 Tonos (3 voces si multáneas) y una señal de ruido (para crear efectos especiales).

MACRO LENGUAJE MUSICAL

La sentencia de ejecución es PLAY, que realiza una funcion similar a la desempenada por DRAW, utilizando un macro lenguaje musical dentro de una expresión STRING El formato es el siguiente

PLAY <exp String para VOZ 1> <exp. String para VOZ 2>, <exp. String para VOZ 3>

<exp String para VOZ N> es una expresion String consistente en comandos musicales de una sola letra Si no se especifica alguna de las 3 voces, el canal correspondiente permanecerá en silencio

Las notas musicales pueden especificarse de dos formas distintas

1°) Utilizando la notación alfabetica convencional

Las notas se representan mediante las letras A-G

C	D	E	F	G	A	B
do	re	mi	fa	sol	la	si

Para indicar un sostenido colocaremos el simbolo # o + a continuacion de la nota Por ejemplo Do sostenido se representara como C # ó C+.

Para indicar un bemol representaremos la nota con el simbolo As D— sera Re bemol.

Unicamente pueden utilizarse los sostenidos y bemoles que corresponden a las tecias negran del piano no siendo validas las siguientes expresiones

- B # (Si sostenido) = C (Do)
- C— (Do bemol) = B (Si)
- E # (Mi sostenido) = F (Fa)
- F— (Fa bemol) = E (Mi)

Para indicar la octava correspondiente utilizaremos el comando **On** (n = 1-8)
Cada octava va de Do a Si (C —B) Si no se especifica octava, se entendera 04.

```
PLAY "C D E F G A B" ← Escala natural
PLAY  CC # DD # EFF # GG # AA # B  ← Escala cromatica
```

```
10 REM$ ACOMPANAMIENTO ROCK $
20 A$     T230O3EG#BO4C#DC#O3BG#
30 B$ = "T230O3AO4C#EF#GF#EC#"
40 C$     T230O3BO4D#F#G#AG#F#D#
50 FOR I = 1 TO 2  PLAY A$  NEXT
60 PLAY B$ : PLAY A$
70 PLAY C$  PLAY B$
80 GOTO 50
```

T 230, indica la velocidad de ejecucion (ver comando T)

2º) Utilizando una notación numérica (Nn)
En lugar de designar las notas con letras (A-G), lo hacemos mediante numeros Nn (n 0 - 96) Esta es una forma de seleccionar las notas sin tener que especificar nombre (A G) octava ni alteraciones (sostenidos y bemoles) La octava 4 correspondera a la siguiente numeracion

NOTA	NOTACION NUMERICA	NOTACION ALFABETICA
DO	36	C
DO #	37	C #
RE	38	D
RE #	39	D #
MI	40	E
FA	41	F
FA #	42	F #
SOL	43	G
SOL #	45	G #
LA	45	A
LA #	46	A #
SI	47	B

Cada octava está formada por 12 notas por lo que para calcular el numero correspondiente a una nota en otra octava, tendremos que sumar/restar 12

Así por ejemplo, DO (Octava 4)   36 (ver Tabla)
DO (Octava 5) = 36 + 12 = 48
DO (Octava 3) = 36 — 12 = 24

```
10 REM $ NOTACION NUMERICA $
20 A $     N36N37N38N39N40N41N42N43N44N45N46N47
30 PLAY A $
```

COMANDOS MUSICALES

Dentro de las expresiones anteriores pueden utilizarse tambien los siguientes comandos

Ln : **Duracion de las notas** (n = 1 - 64)
La duración sera 1/n

Ln	Duración
L1	1
L2	1/2
L3	1/3
L4	1/4
.	.
	.
L64	1/64

Si dentro de una expresión queremos variar la duración de una nota, lo haremos colocando el valor al lado de ésta. Así por ejemplo: A16 equivale a L16A, pero L16AB es distinto de A16B (debería ser A16B16).

Rn **Pausa** (n – 1-64). Se utiliza para simular los "silencios". La duración se calcula del mismo modo que para Ln.

. **Punto**. Realiza la operación del "puntillo" (la duración de la nota precedente se multiplica por 3/2). Puede aplicarse también a la pausa (Rn).

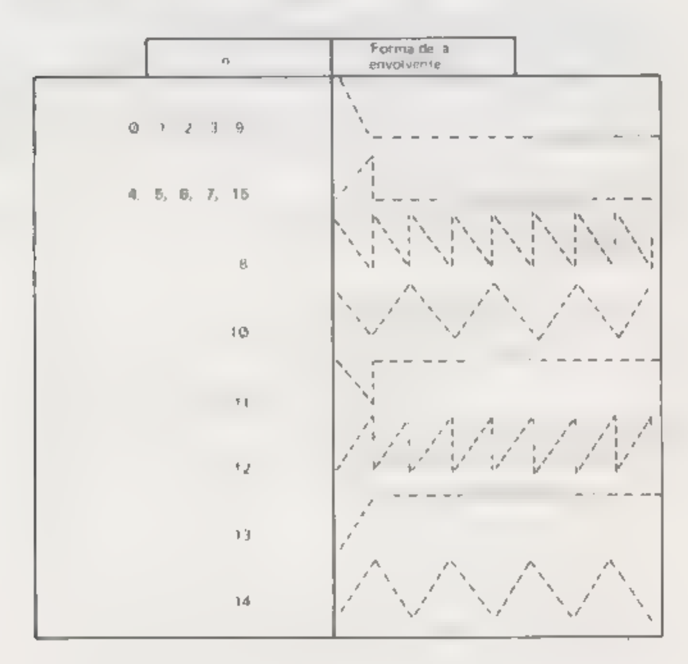
Ejemplo: G2. = GGG (duración de G = 2 x 3/2 = 6/2 = 3).

Tn **Tiempo**: velocidad de ejecución (n 32 – 255) Si no se indica, se entenderá T12Ø.

Vn **Volúmen**: volúmen de salida (n = Ø - 15)

Mn **Modulación**: período de la envolvente (n = Ø — 65.535).

Sn **Forma**: forma de la envolvente (n Ø – 15):



X <variable>: Ejecuta el String especificado.

En todos estos comandos, el argumento n puede ser constante, como por ejemplo 17, ó variable. En tal caso, lo escribiremos de la forma: " = < variable >; ", donde variable es el nombre de una variable (X por ejemplo).

Ejemplo: Este programa, nos pedirá que le entremos el valor del período de la envolvente y nos ejecutará una pequeña melodía con las distintas formas de envolvente (ver Tabla II).

```
1Ø INPUT "PERIODO:"; B
2Ø A$ = T23Ø03EG#B04C#DC#03BG#"
3Ø FOR A = 3 TO 15
4Ø PLAY "M=B;" : PLAY "S=A;"
5Ø PLAY A$
6Ø NEXT
```

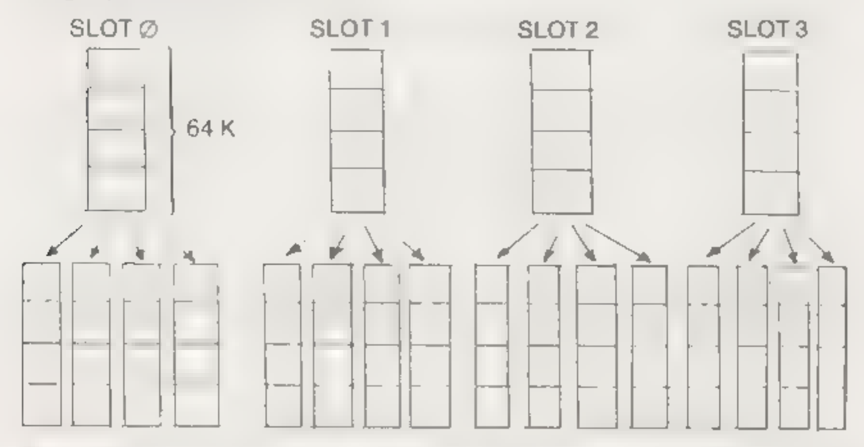


ANEXO 3: INFORMACION TECNICA

1. MAPA DE MEMORIA

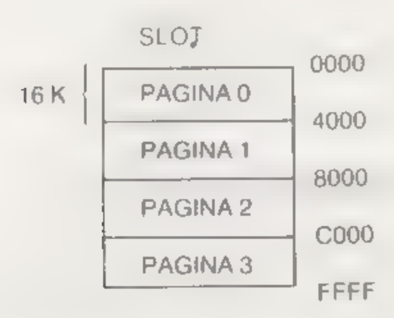
El Mapa de memoria de un ordenador MSX esta estructurado en cuatro bloques de 64K denomindos SLOTS primarios con lo que se dispone de un espacio fisico en memoria de 256K

Cada uno de estos SLOTS puede ampliarse hasta un maximo de cuatro (SLOTS secundarios) disponiendo asi de un espacio en memoria de 1 Megabyte. (espacio fisico total)



El microprocesador Z80 tiene 16 bits de direccionamiento (BUS de direcciones) por lo que directamente solo puede acceder a 64K (2^{16} = 65 536 = 64K) Para permitir el acceso a todo el espacio de memoria disponible cada SLOT se divide en cuatro paginas logicas de 16K y mediante el registro selector de SLOT (PORT A del CI-8255) se transforma el espacio fisico de

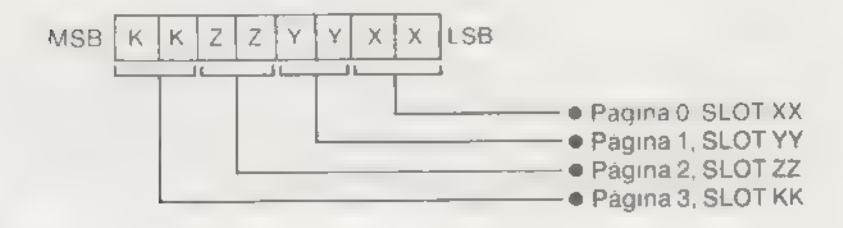
memoria en un bloque logico de 64K formado por pag nas de 16K de los distintos SLOTS De esta forma la memoria fisica esta s empre situada en e espacio de direccionamiento de la CPU



La ROM del BASIC MSX (BIOS + Interprete BASIC MSX) ocupa as pag nas 0 y 1 de SLOT 0 por o que la maxima memoria RAM direccionable, trabajando en BASIC es de 32KBytes De estas 32KB el sistema se reserva 4K para almacenamiento de variables y direccionamiento de os dispositivos de Entrada Saida quedando disponibles para e usuario 28 9KB

El BASIC MSX ut liza la mayor area de RAM contigua d sponible nsta ada entre las paginas 2 y 3 por o que si disponemos de un SLOT con una pagina de RAM y otro con dos (d reccionables en BASIC) automaticamente al in c alizarse el Sistema este seleccionara como memor a RAM de trabajo a s tuada en las dos paginas contiguas quedando la de otro SLOT inutilizada

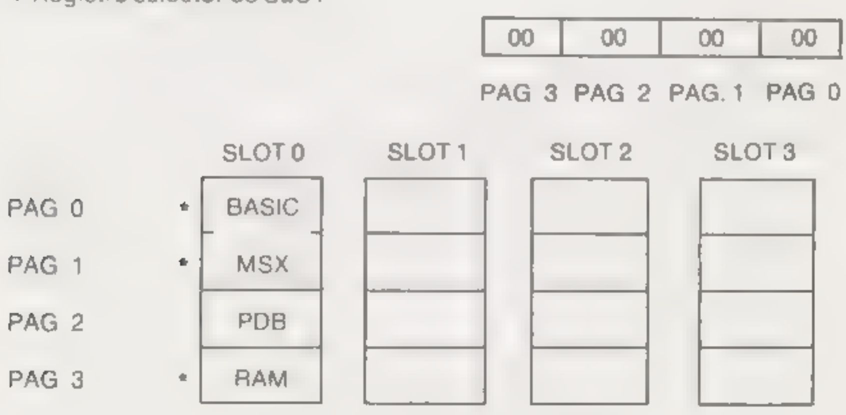
La selecc on de SLOTS para cada pagina esta determinada por el conten do del reg stro selector de SLOT (PORT A del CI 8255)



Con el valor de este registro podemos hacer un esquema del Mapa de memoria en cada configuración

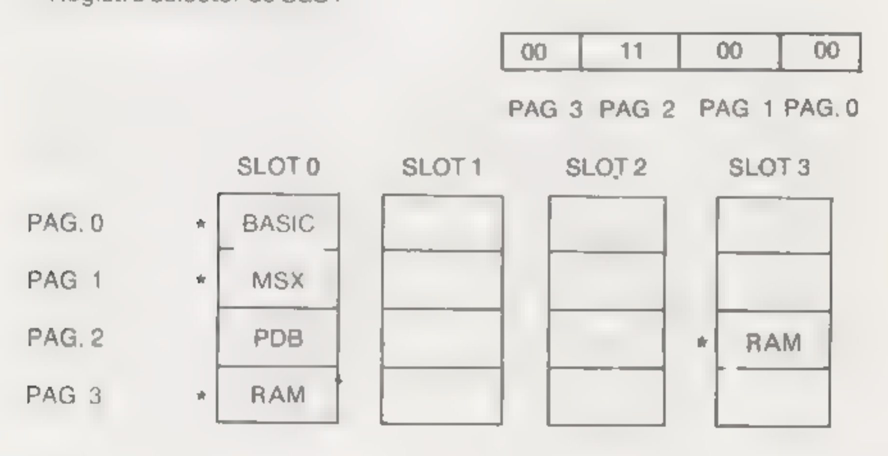
1°) HIT BIT 55

● Registro selector de SLOT



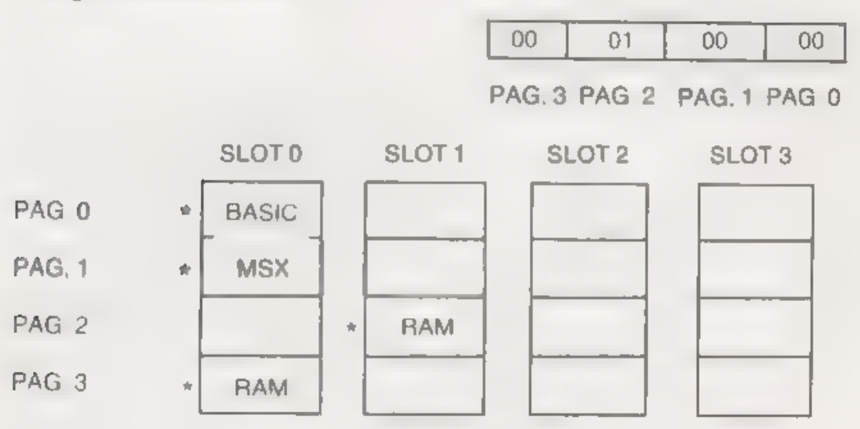
2°) HIT BIT 55 + HBM 16 (conector posterior HIT BIT)

● Registro selector de SLOT



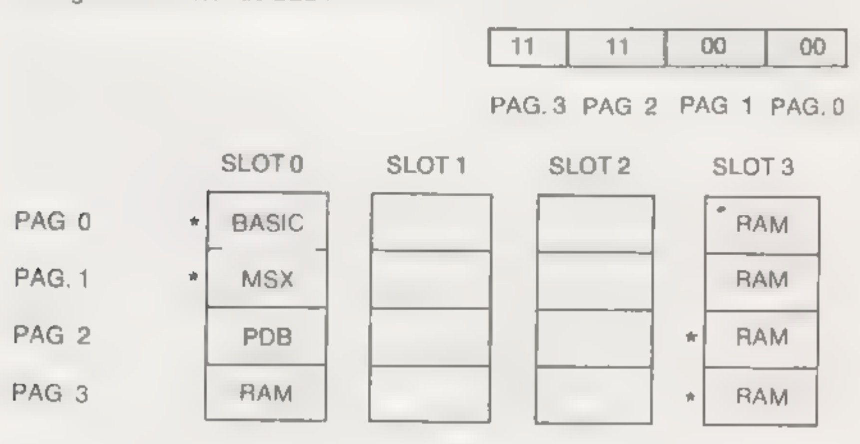
3°) HIT BIT 55 + HBM 16 (conector superior)

● Registro selector de SLOT



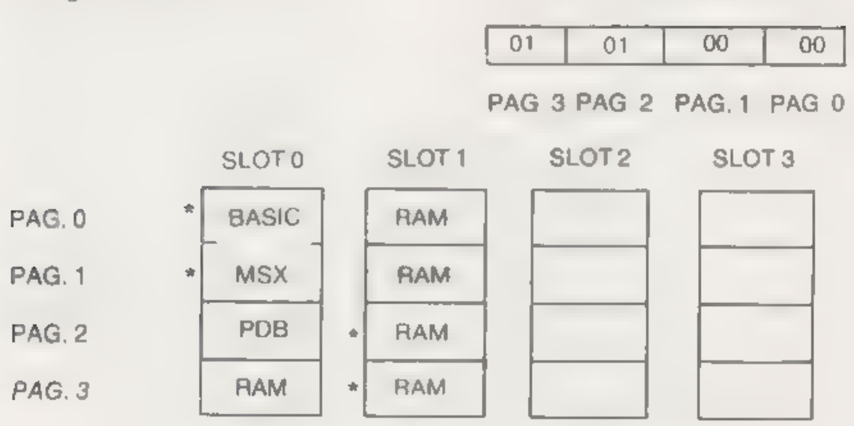
4°) HIT BIT 55 + HBM 64 (conector posterior)

● Registro selector de SLOT·



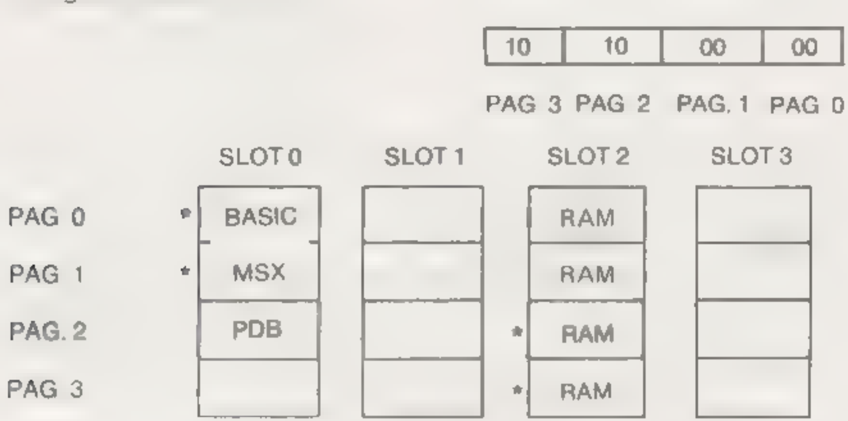
5°) HIT BIT 55 + HBM 64 (conector superior)

● Registro selector de SLOT



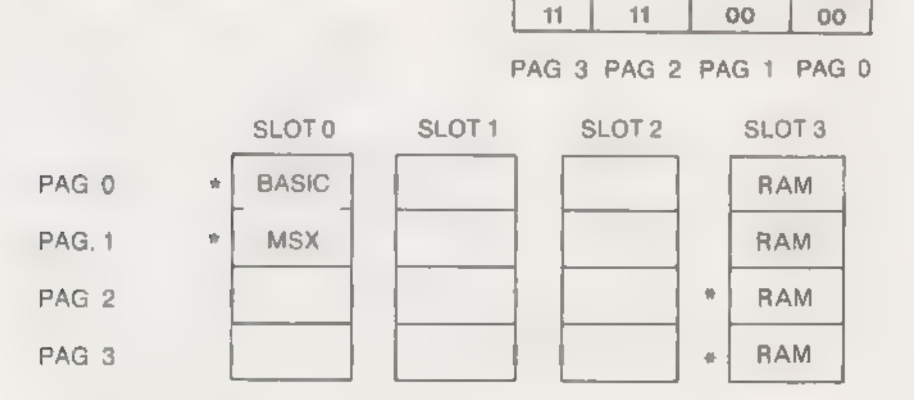
6°) *HIT BIT 75*

● Registro selector de SLOT



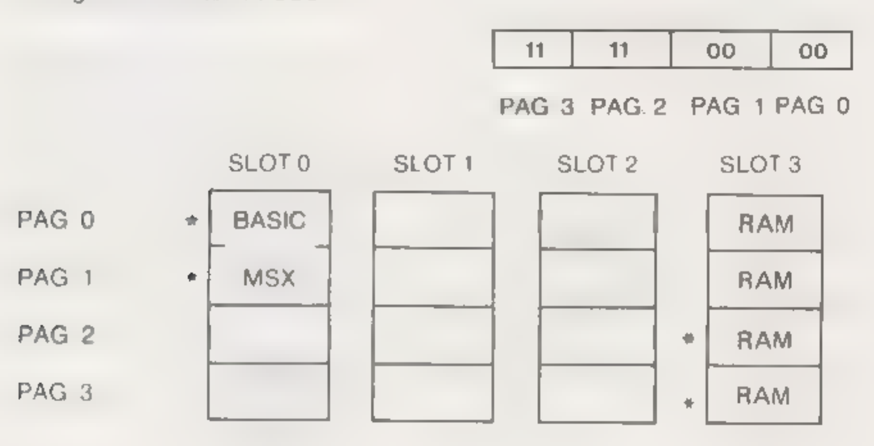
7º) HIT BIT 10P

● Registro selector de SLOT



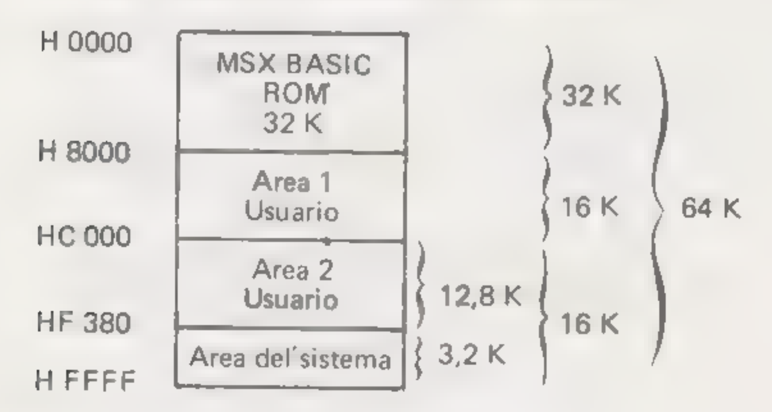
8º) HIT BIT 20P

● Registro selector de SLOT



MAPA DE MEMORIA

I. Disposición general trabajando en BÁSIC-MSX:



II. Area disponible por el usuario:

H 8000
(HC 000)

Area de Programa

Area de variables

Area de arrays

Area Libre

Area del stack

Area de strings

Bloque de control de Ficheros

H 37 F

Con 64 K RAM — 28,8 K

Con 16 K RAM → 12,8 K

CONFIGURACION DEL AREA DE USUARIO

Trabajando en BASIC la RAM de usuario tiene la siguiente estructura

Dirección	Área	Descripción
8000 (C000)	AREA DE PROGRAMA	• Almacena programas BASIC
	AREA DE VARIABLES	• Almacena datos numericos aislados a variables y punteros de los datos alfanumericos
	AREA DE VARIABLES DE MATRIZ	• Almacena datos numéricos asignados a variables de matriz y punteros de los datos alfanuméricos
	AREA LIBRE	• Zona de memoria libre disponible. El tamaño de esta zona puede conocerse utilizando la funcion FRE
	AREA DE STACK	• Almacena las direcciones de retorno para los comandos de bifurcación (FOR NEXT, GOSUB...)
	AREA DE VARIABLES ALFANUMERICAS	• Almacena "strings" (cadenas de caracteres) asignados a variables alfanuméricas El tamaño de esta zona, puede ser definido por el usuario mediante el comando CLEAR Inicialmente, se reservan 200 Bytes para esta área
F37F	BLOQUE DE CONTROL DE FICHEROS	• Utilizado por los Ficheros de Entrada/Salida El tamaño de este bloque, corresponde al numero de Ficheros especificados en el comando MAXFILES

OBSERVACIONES

- Normalmente los SLOTS 0 y 2 son internos al sistema
  - — SLOT 0, Página 0 y Página 1: BASIC MSX
  - SLOT 0 Pagina 2 PERSONAL DATA BANK

- La direccion de memoria FFFF de los SLOTS «primarios» contiene la situacion del Registro de seleccion de SLOT «secundario». El valor real de este Registro es el complemento de su valor actual

- El simbolo ' indica las paginas direccionadas en BASIC que corresponden a las indicadas en el Registro selector de SLOT

- Todos los ordenadores MSX disponen de 16K de memoria RAM de Video (VRAM) que se utiliza exclusivamente para la visualizacion en pantalla por lo que no se incluyen en el Mapa de Memoria.

- EL Sistema Operativo de Disco MSX DOS requiere 64K de memoria de RAM

MAPA DE ENTRADA/SALIDA

DIRECCION

Dirección	
FF–E0	Area reservada
E0–D8	
D8–D0	Controlador Lector diskettes
D0–B0	Area Reservada
B0–A8	Interface Programable (PPI)
A8–A0	Procesador de sonido
A0–98	Procesador de video
98–90	Impresora
90–88	Area Reservada
88–80	RS 232C
80–00	Area Reservada

Dirección	L/E	Aplicación	OBSERVACIONES
80	L	Lectura Datos	INTERFACE RS 232C
	E	Ent. Datos	
81	L	Estado	
	E	Est. Modo	
90	L	Lectura Status (B0)	IMPRESORA
	E	Salida Strobe (B1)	
91	E	Ent. Datos	
98	L	Lectura datos en Ram de video	PROCESADOR DE VIDEO 9928
	E	Ent. datos en Ram de video	
99	L	Estado	
	E	Establecimiento Mandatos/Direcc	
A0	E	Mantenimiento de Dirección	PROCESADOR DE SONIDO AY-3-8910
A1	E	Ent. datos	
A2	L	Lectura datos	
A8	L	Lectura datos Port A	PPI 8255
	E	Ent. datos Port A	
A9	L	Lectura datos Port B	
	E	Ent. datos Port B	
AA	L	Lectura datos Port C	
	E	Ent. datos Port C	
AB	E	Est. Modo trabajo	

L = LECTURA
E = ESCRITURA

DISTRIBUCION DE LOS PORTS DE E/S DEL PPI (8255)

PORT	BIT	E/S	SENAL	APLICACION
A	0 1	S S	CS0L CS0H	Selección de Slot en Página 0
	2 3	S S	CS1L CS1H	Selección de Slot en Página 1
	4 5	S S	CS2L CS2H	Selección de Slot en Página 2
	6 7	S S	CS3L CS3H	Selección de Slot en Página 3
B	0 7	E E		Teclado
C	0	S	KB0	Teclado
	1	S	KB1	
	2	S	KB2	
	3	S	KB3	
	4	S	CASON	Control de cassette (0 ON)
	5	S	CASW	Escritura cassette
	6	S	CAPS	Lampara Caps (0 ON)
	7	S	SOUND	Salida de sonido (cuando está controlado por Soft)

Página 0. 0000 - 3FFF
Página 1: 4000 - 7FFF
Página 2: 8000 - BFFF
Página 3: C000 - FFFF
E = Entrada
S = Salida

DISTRIBUCION DE LOS PORTS DE E/S DEL PROCESADOR DE SONIDO

PORT	BIT	E/S	APLICACION	OBSERVACIONES
A	0	E	JS1 - PIN 1 *1 JS2 - PIN 1 *2	Adelante
	1	E	JS1 - PIN 2 *1 JS2 - PIN 2 *2	Atras
	2	E	JS1 - PIN 3 *1 JS2 - PIN 3 *2	Izquierda
	3	E	JS1 - PIN 4 *1 JS2 - PIN 4 *2	Derecha
	4	E	JS1 - PIN 6 *1 JS2 - PIN 6 *2	Boton de disparo 1
	5	E	JS1 - PIN 7 *1 JS2 - PIN 7 *2	Boton de disparo 2
	6	E	Seleccion caracteres del teclado	Solo versión Japonesa
	7	E	CSAR	Lectura cassette
B	0	S	JS1 - PIN 6 *3	
	1	S	JS1 - PIN 7 *3	
	2	S	JS2 - PIN 6 *3	
	3	S	JS2 - PIN 7 *3	
	4	S	JS1 - PIN 7	
	5	S	JS2 - PIN 8	
	6	S	Seleccion entrada Port A	Seleccion JS1/JS2
	7	S	Klamp (lámpara Kana)	Sólo versión japonesa

JS1 - JOYSTICK 1
JS2 - JOYSTICK 2
E - ENTRADA
S - SALIDA
*1 - ACTIVADO CUANDO EL BIT 6 DEL PORT B ES 0
*2 - ACTIVADO CUANDO EL BIT 6 DEL PORT B ES 1
*3 SI EL PORT B NO SE UTILIZA PARA SALIDA ESTE BIT DEBE SER 1

2. PROCESADOR DE VIDEO (VDP)

El ordenador incorpora el circuito integrado TMS 9928 A que actua como procesador de imagen de video Paralelamente va incorporada una memoria adicional de memoria RAM de 16 K donde se almacena la informacion de las posiciones en la pantalla junto con la definicion de cada caracter

En cada tipo de pantalla la informacion queda almacenada en distintas partes de la memoria Mediante las funciones VPEEK y VPOKE podemos examinar y alterar dichas posiciones

La funcion BASE(N) se utiliza para leer o escribir una direccion base de la tabla del procesador de visualizacion (VDP) El contenido de los registros y la direccion base de la tabla del TMS 9928 A que es el contenido de la pantalla podra modificarse directamente utilizando una variable BASE y una variable VDP Es necesario conocer adecuadamente el TMS 9928 A ya que podria alterar la visualizacion normal de la pantalla

A continuacion se detallan los valores posibles de la funcion BASE segun el tipo de pantalla utilizada

Valor de N	Tabla
0	Tabla de nombres de patrones del modo de texto de 40 caracteres × 24 líneas
2	Tabla del generador de patrones del modo de texto de 40 caracteres × 24 lineas
5	Tabla de nombres de patrones del modo de texto de 32 caracteres × 24 lineas
6	Tabla de colores del modo de texto de 32 caracteres × 24 lineas.
7	Tabla del generador de patrones del modo de texto de 32 × 24 caracteres
8	Tabla de atributos de figuras moviles del modo de texto de 32 × 24 caracteres
9	Tabla de patrones de figura movil del modo de texto de 32 × 24 caracteres
10	Tabla de nombres de patrones del modo de gráficos de gran definición.
11	Tabla de colores del modo de gráficos de gran definición
12	Tabla del generador de patrones del modo de gráficos de gran definición.
13	Tabla de atributos de figuras moviles del modo de gráficos de gran definición
14	Tabla de patrones de figura movil del modo de graficos de gran definición.
15	Tabla de nombres de patrones del modo multicolor
17	Tabla del generador de patrones del modo multicolor
18	Tabla de atributos de figuras moviles del modo multicolor
19	Tabla de patrones de figura movil del modo multicolor

N = 1, 3, 4, y 16 no se utilizan

En cada tipo de pantalla (SCREEN) se manejan una serie de tablas cuyos detalles se especifican a continuación.

SCREEN 0

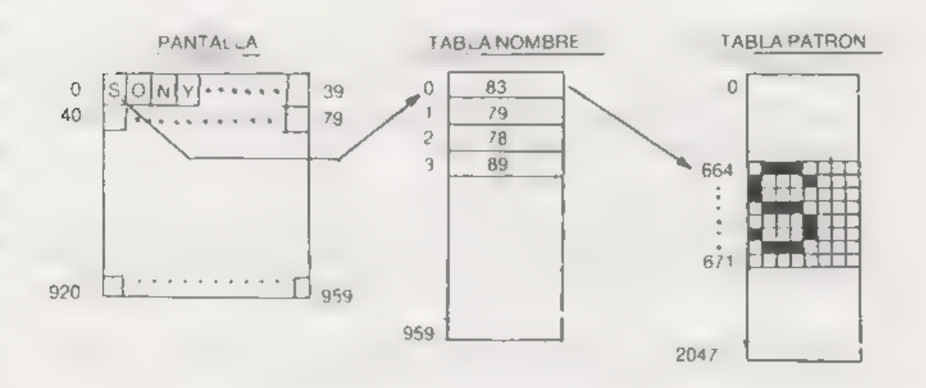
SCREEN 0 solo puede contener caracteres (Maximo 40 × 24), un color de pantalla, no es posible utilizar SPRITES y el color de los bordes es el de fondo.

La pantalla queda dividida en 40 lineas verticales y 24 lineas horizontales, por lo que disponemos de 960 posiciones

Existen dos tablas para el manejo y almacenamiento de los caracteres La tabla NOMBRE contiene el codigo de caracter de cada posicion de pantalla La tabla PATRON contiene el diseno de los diferentes caracteres Cada posicion de la tabla NOMBRE tiene reservados 8 octetos de la tabla PATRON

El inicio de la tabla NOMBRE viene dado por la variable BASE (0) y el de la tabla PATRON por BASE (2)

Imaginemos que en la primera linea de la pantalla tenemos la palabra SONY



La primera posicion de la pantalla es la letra S cuyo codigo ASCII (83) está reflejado en la primera posicion de la tabla NOMBRE Este mismo codigo (83), multiplicado por ocho da la primera posición de la tabla PATRON, donde se encuentra reflejada la forma de este carácter La estructura de este carácter se encuentra en las 8 posiciones consecutivas de la tabla PATRON (664-671).

EJEMPLO

S. ejecuta el s gu ente ejemp o podra comprobar e conten do de la tabla PATRON

```
 10 SCREEN 0  WIDTH 40  D$  STRING$ (8   0  )
 20 PRINT  SONY
 30 FOR K  0 TO 3
 40 I   VPEEK (BASE (0) + K)
 50 Y  I *8  Z  Y + 8  J  J + 9  T  0
 60 FOR X  Y TO Z-1
 70 LOCATE J 5  7 + T
 80 PRINT RIGHT$ (D$ + BIN$ (VPEEK (BASE (2) + X)), 8)
 90 T  T + 1
100 NEXT X  K
```

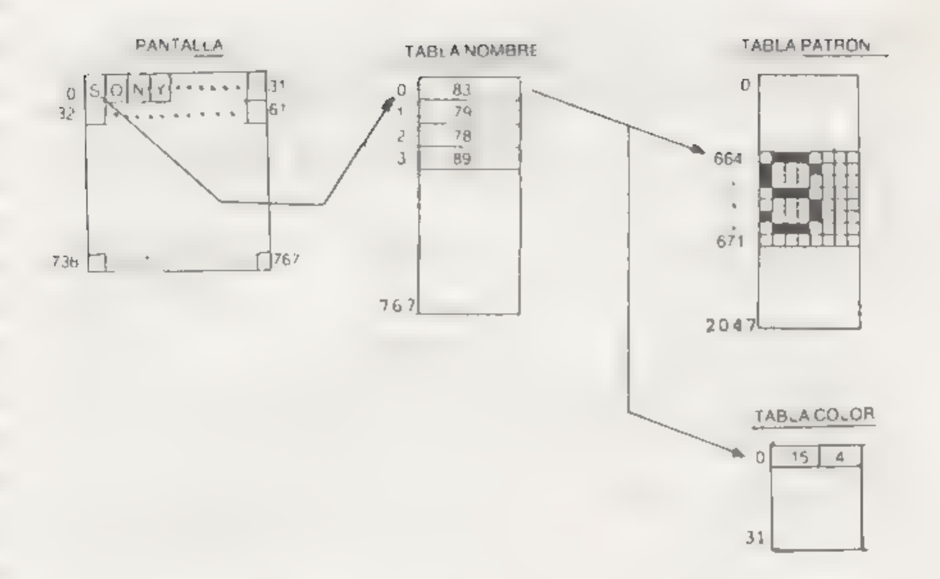
SCREEN 1

En SCREEN 1 cada 8 caracteres pueden presentarse en diferente color puede manejar SPRITES el maximo numero de caracteres es de 32 × 24

La pantalla queda d vidida en 32 lineas verticales y 24 horizontales por lo que disponemos de 768 posiciones.

En este caso se manejan tres tablas en la memoria RAM de video La tabla NOMBRE y PATRON funcionan de forma similar a la descrita en SCREEN 0 La tercera tabla tabla COLOR es la correspondiente al color de cada caracter y de su fondo de la tabla NOMBRE

E inic o de a tabla NOMBRE viene determinado por la variable BASE (5) la tabla PATRON por BASE (7) y la de color por BASE (6)



Los colores de fondo y de los caracteres se pueden modificar independientemente con la sentencia COLOR Ademas pueden presentarse diferentes colores a la vez

BASE (6) indica el principio de la tabla COLOR de 32 bytes Cada byte esta divid do en dos cuartetos El primer cuarteto indica el color de los ocho primeros caracteres consecutivos de la pantalla y el segundo cuarteto el color de fondo

SCREEN 2

En SCREEN 2 pueden presentarse graficos de alta resolucion y SPRITES, texto en pantalla grafica 256 × 192 puntos de resolucion y cada 8 puntos tiene su propio color de frente y fondo

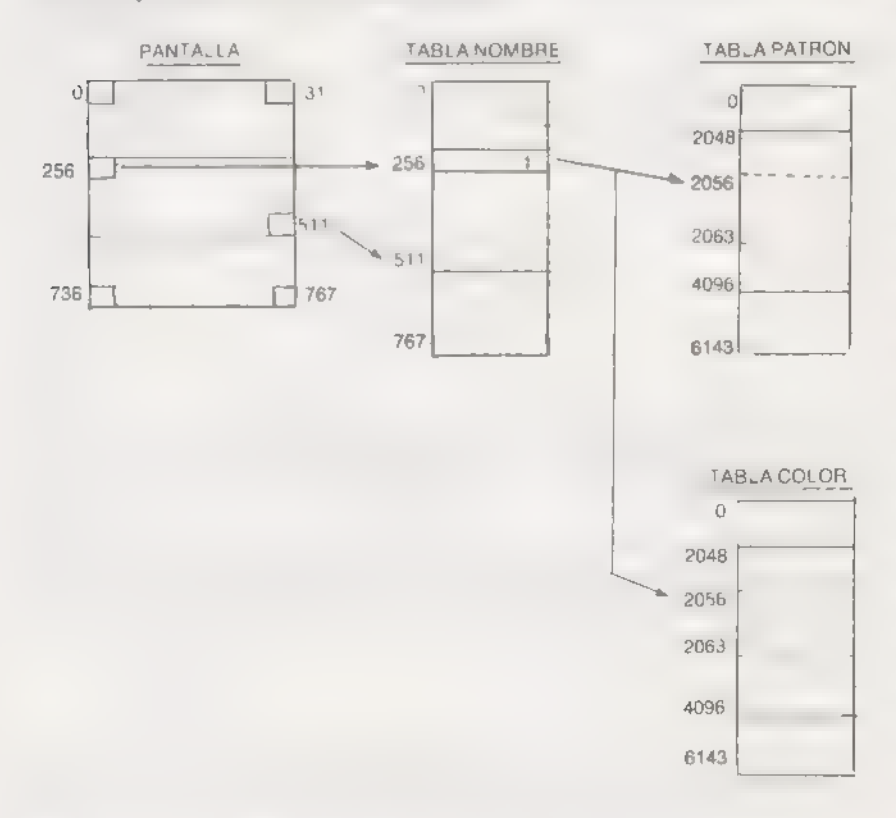
La pantalla queda divid da en 24 lineas horizontales y 32 vertica es configurando un tota de 768 posic ones de pantalla Cada posicion está subdividida en una malla de 8×8 puntos.

Se conserva el numero de tablas aunque varia su longitud La tabla NOMBRE contiene el codigo de caracter de cada posicion correlativa de la pantalla

El codigo de caracter que figura en la tabla NOMBRE senala la posición del tipo y color del caracter en la tabla PATRON y tabla COLOR respectivamente

Cada caracter de la tabla NOMBRE esta constituido por una matriz de 8 × 8 puntos en la tabla PATRON. Por lo tanto la tabla PATRON estara compuesta de 768 posiciones de 8 bytes resultando 6144 bytes.

Para cada posición de la tabla PATRON existe el color correspondiente del caracter y de su fondo en la tabla COLOR.



El inicio de la tabla NOMBRE viene determinado por la variable BASE (10) la tabla PATRON por BASE (12) y la tabla COLOR por BASE (11).

SCREEN 3

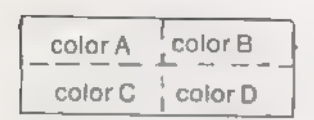
En SCREEN 3 pueden presentarse graficos de baja resolución y texto en pantalla gráfica en modo multicolor.

Este modo proporciona una definición de 64 × 48 bloques de color. Cada bloque contiene 4 × 4 puntos. El color de cada uno de los cuatro puntos

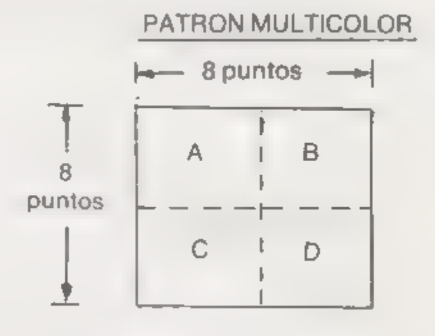
puede ser uno de los 16 colores existentes Consecuentemente, todos los colores pueden ser usados s multaneamente en el modo SCREEN 3 Los planos SPRITES estan permitidos en este modo

En este tipo de pantalla existen dos tablas La tabla NOMBRE es la m sma que para SCREEN 2 cons stiendo en 768 posic ones El color esta ahora definido en la tabla PATRON El codigo de caracter de la tabla NOMBRE senala un segmento de 8 bytes de VRAM en la tabla PATRON

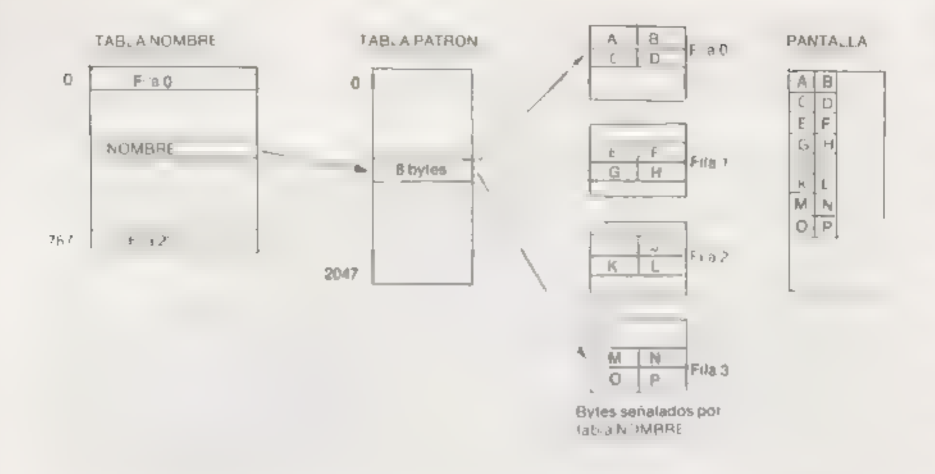
Solo 2 bytes de los 8 que forman el segmento son usados para especificar la imagen de pantalla Estos 2 bytes especifican 4 colores y cada color ocupa un area de 4 · 4 puntos Los cuatro primeros bits (MSB) del primer byte definen el color del bloque izquierdo superior del patrón multicolor, los 4 bits restantes definen el color del bloque superior derecho El segundo bytes define de forma parecida el bloque inferior izquierdo y derecho del patron mu tico or Los 2 bytes crean un mapa de 8 · 8 puntos de patron multicolor



2 Bytes de la Tabla
Generador de Patrón



La pos cion de los 2 bytes dentro del segmento de 8 bytes senalado por el codigo de caracter de la tabla NOMBRE depende de la posicion que aparece el caracter en la pantalla Para nombres en la fila superior (fila 0) los 2 bytes son los 2 primeros dentro de los grupos de segmentos de 8 bytes senalados por el codigo de caracter de la tabla NOMBRE La siguiente fila de caracteres (fila 1) ut liza los bytes 3 y 4 de los segmentos de 8 bytes La sigu ente fila ut liza los bytes 5 y 6 mientras que la ultima fila uti iza los bytes 7 y 8 Esta ser e se repite para el resto de pantalla



Cuando se utiliza este modo 768 bytes son utilizados por la tab a NOMBRE y 1536 bytes son usados para la informacion de color en la tabla PATRON (24 filas × 32 columnas × 8 bytes/posicion patron)

3. CONECTORES

1. CASSETTE:

PIN	SENAL	E/S	CONECTOR
1	MASA	—	
2	MASA	—	
3	MASA	—	
4	SALIDA	S	
5	ENTRADA	E	
6	REMOTO +	S	
7	REMOTO –	S	
8	MASA	—	

E = ENTRADA
S = SALIDA

2. AUDIO/VIDEO

PIN	SEÑAL	E/S	CONECTOR
1	+ 12V	—	
2	MASA	—	
3	+ 12V	—	
4	VIDEO	S	
5	AUDIO	S	
6	AUDIO	S	

3. JOYSTICK

PIN	SEÑAL	E/S	CONECTOR
1	ADELANTE	E	
2	ATRAS	E	
3	IZQUIERDA	E	
4	DERECHA	E	
5	+ 5 V	—	
6	B DISPARO 1	E/S	
7	B DISPARO 2	S	
8	SALIDA	S	
9	MASA	—	

4. IMPRESORA:

PIN	SEÑAL	PIN	SEÑAL	CONECTOR
1	PSTB	8	DATO 6	
2	DATO 0	9	DATO 7	
3	DATO 1	10	SIN CONEX	
4	DATO 2	11	BUSY	
5	DATO 3	12	SIN CONEX.	
6	DATO 4	13	SIN CONEX.	
7	DATO 5	14	MASA	

5. CARTUCHO:

Pin 49 1

Pin 50 2

PIN	SEÑAL	E/S	PIN	SEÑAL	E/S
1	CS1	S	2	CS2	S
3	CS12	S	4	SLTSL	S
5	Reservado	—	6	RFSH	S
7	WAIT	E	8	INT	E
9	M1	S	10	BUSDIR	E
11	IORQ	S	12	MERQ	S
13	WR	S	14	RD	S
15	RESET	S	16	Reservado	—
17	A9	S	18	A15	S
19	A11	S	20	A10	S
21	A7	S	22	A6	S
23	A12	S	24	A8	S
25	A14	S	26	A13	S
27	A1	S	28	A0	S
29	A3	S	30	A2	S
31	A5	S	32	A4	S
33	D1	E/S	34	D0	E/S
35	D3	E/S	36	D2	E/S
37	D5	E/S	38	D4	E/S
39	D7	E/S	40	D6	E/S
41	Masa	—	42	Reloj	S
43	Masa	—	44	SW1	—
45	+ 5V	—	46	SW2	—
47	+ 5V	—	48	+ 12V	—
49	SOUNDIN	E	50	— 12V	—

PIN	NOMBRE	CONTENIDO
1	CS1	Señal de selección de direcciones ROM 4000/7FFF
2	CS2	Señal de selección de direcciones ROM 8000/BFFF
3	CS12	Señal de selección de direcciones ROM 4000/BFFF (para 256 K ROM)
4	SLTSL	Señal de selección de SLOT
5	Reservado	Línea de señal reservada (uso inhibido)
6	RFSH	Ciclo de señal de refresco
7	WAIT	Señal de requerimiento de espera de la CPU
8	INT	Señal de requerimiento de interrupción de la CPU
9	M1	Señal de ciclo de fetch de CPU
10	BUSDIR	Esta señal controla la dirección del buffer del bus de datos externos. Los cartuchos son seleccionados y el niel "L" es sacado de cada cartucho en el momento de transmisión de datos
11	IORQ	Señal de requerimiento E/S
12	MERQ	Señal de requerimiento de memoria
13	WR	Señal tiempo escritura
14	RD	Señal de tiempo de lectura
15	RESET	Señal de RESET del sistema
16	RESERVADO	Línea de señal reservada (uso inhibido)
17 32	AO A15	Señales del bus de direcciones.
33 40	DO D7	Señales del bus de datos
41	MASA	Señal de Masa
42	RELOJ	Reloj CPU 3.58 MHZ.
43	MASA	Señal de Masa
44 46	SW1 SW2	Para protección de Inserción/Ordenación
45 47	+ 5V	Tensión + 5V
48	+ 12 V	Tensión + 12V
49	SOUNDIN	Señal de entrada de sonido (- 5 dbm)
50	- 12 V	Tensión 12 V

6. RGB:

PIN	SEÑAL	PIN	SEÑAL	CONECTOR
1	SAL AUDIO D	12		
A 2	ENT AUDIO D	13	MASA	
3	SAL AUDIO I	14		
4	MASA	15	COLOR ROJO	
5	MASA	16	BLANKING	
A 6	ENT AUDIO I	17	MASA	
7	COLOR AZUL	18	MASA	
8	—	19	SAL VIDEO	
9	MASA	20	—	
10	—	21	—	
11	COLOR VERDE			

19: masa. video
20: ent. video



ANEXO 4: PROGRAMAS

PROGRAMA DEMOSTRACION DE LOS COMANDOS GRAFICOS UTILIZADOS EN EL BASIC MSX

```
1 REM ** COMANDOS GRAFICOS **
10 COLOR 15,1,1
15 '*** ALTA RESOLUCION -->SCREEN 2
20 SCREEN 2
25 '*** PUNTO ***
30 PSET(32,32),15
35 '*** LINEA ***
40 LINE(76,32)-(116,32),8
45 '*** RECTANGULO ***
50 LINE(135,12)-(180,52),12,B
55 '*** PINTAR UN RECTANGULO ***
60 LINE(204,12)-(244,52),8,BF
65 '*** CIRCULO ***
70 CIRCLE(32,96),20,5
75 '*** PINTAR UN CIRCULO ***
80 CIRCLE(96,96),20,4
90 PAINT(96,96),4
95 '*** ELIPSE  HORIZONTAL ***
100 CIRCLE(160,96),20,15,,,.5
105 '***PINTAR UNA ELIPSE HORIZONTAL***
110 CIRCLE(224,96),20,13,,,.5
120 PAINT(224,96),13
125 '*** ELIPSE VERTICAL ***
130 CIRCLE(32,160),20,2,,,2
```

```
130 '*** PINTAR [illegible] ***
140 CIRCLE(96,160),20,11...2
150 PAINT(96,160),11
155 '*** MACRO LENGUAJE GRAFICO ***
160 DR[illegible] bm140,160 [illegible] e2[illegible] 10e[illegible]
r5e5f10r[illegible]0
170 GOTO 170
```

PROGRAMA DE GRAFICOS: SIMULACION REBOTES DE UNA PELOTA

```
10 '**REBOTES DE UNA PELOTA**
20 SCREEN 2,0:COLOR 15,1,1
30 C[illegible]S:A RND([illegible]TIME[illegible]:T [illegible]INT(RND(1)*14+[illegible]):O
PEN GRP: FOR OUTPUT AS#1:PRESET(20,10):C
OLOR T:PRINT#1, REBOTES DE UNA PELOTA
40 X=10:Y=10:A=4:V=0:C=0
50 CIRCLE(X,Y),5,C,,,1.2
60 ON INTERVAL=5 GOSUB 100
70 INTERVAL ON
80 IF X<256 THEN 80
90 INTERVAL OFF:CLOSE:GOTO 30
100 CIRCLE(X,Y),5,T,,:1.2
110 V=V+A
120 X=X+5:Y=Y+V
130 IF Y>180 THEN Y 180*2 [illegible] X,R,BEEP

140 CIRCLE(X,Y),5,T
150 RETURN
```

PROGRAMA DE GRAFICOS: CUADROS ABSTRACTOS

```
10 REM *** CUADROS ABSTRACTOS ***
20 SCREEN 2:COLOR 15,1,1:CLS
22 A=RND(-TIME)
25 FOR I=1 TO 75
```

```
27 C=INT(RND(1)*14)+2
30 X1=INT(RND(1)*255)
40 Y1=INT(RND(1)*192)
50 X2=INT(RND(1)*255)
60 Y2=INT(RND(1)*192)
70 LINE(X1,Y1)-(X2,Y2),C,BF
80 NEXT
90 FOR I=1 TO 2000:NEXT:GOTO 20
```

PROGRAMA DE GRAFICOS: CIRCULOS DE COLORES

```
10 COLOR 15,1,1
20 SCREEN2:A=RND(-TIME)
30 FOR X=0 TO 2*3.14159 STEP .104719666#

40 CIRCLE(124+40*COS(X),106+40*SIN(X)),2
0,INT(RND(1)*15)+1,,,1
50 NEXT
60 GOTO 60
```

PROGRAMA DE GRAFICOS: FLORES

```
10 '** GRAFICOS ALEATORIOS **
20 COLOR 15,1,1:SCREEN 2:A=RND(-TIME):
30 R1=5:R2=20:Z=3.14/180
40 X0=INT(255*RND(1)):Y0=INT(192*RND(1))
:C=INT(14*RND(1))+2
50 FOR T=0 TO 360 STEP 20
60 S=T*Z
70 Y=SIN(S):X=COS(S)
80 LINE(R1*X+X0,R1*Y+Y0)-(R2*X+X0,R2*Y+Y
0),
90 NEXT T
100 GOTO 40
```

PROGRAMA DE GRAFICOS: DIBUJO DE CIRCULOS DE COLORES ALEATORIOS EN FORMA SENOIDAL

```
5 '*** GRAFICOS ALEATORIOS ***
10 COLOR 15,1,1
20 SCREEN2:C=2:A=RND(-TIME)
30 FOR X=0 TO 6.28 STEP.157
40 COLOR C:C=C+1:IF C=16 THEN C=2
50 Y=SIN(X)*80+96:A=INT(RND(1)*96)
60 Z=COS(X+(3.1415/2))*80+96
70 CIRCLE(X*30+20,Y),12,,,,.5:PLAY"N=A;"

80 CIRCLE(X*30+20,Z),12,,,,.5
90 NEXT
100 GOTO 30
```

PROGRAMA DE GRAFICOS: 'ESTRELLAS'

```
10 REM $ GRAFICOS:PSET $
20 REM 'CIELO ESTRELLADO'
30 A=RND( TIME):COLOR 15,1,1:SCREEN 2,1:
FOR I=0 TO 200
40 X=INT(RND(1)*255)+1
50 Y=INT(RND(1)*192)+1
60 PSET(X,Y),INT(RND(1)*14)+2
70 NEXT I
80 FOR I=1 TO 2000 :NEXT I
[illegible]
[illegible]
[illegible]
(&B01111110)+CHR$(&B00100100)
100 PUT SPRITE 0,(120,90)
110 FOR Q=1 TO 200:X=INT(RND(1)*20)*(-1)
[illegible]
120 PUT SPRITE 0,STEP(X,Y),5:BEEP
130 FOR T=1 TO 70:NEXT
140 NEXT Q:FOR I=1 TO 2000:NEXT:CLS:GOTO
 30
```

PROGRAMA DEMOSTRACION DE GRAFICOS. 1.

```
10 REM **** GRAFICOS ****
20 ' INICIALIZACION
30 XC=128:YC=96
40 R=8
50 SP=6
60 RATIO=1.5
70 RX [illegible]
80 [illegible] 1
90 PI=3.14159:FLAG=1
100 XB=0:YB=0:XE=0:YE=0:XM=0:YM=0
110 DIM X(6),Y(6),A(6)
120 A(0)=0:A(1)=-15:A(2)=0:A(3)=-15:A(4)
=0:A(5)=-15
130 FOR NO=0 TO 5
140 A(NO)=A(NO)/180*PI
150 NEXT NO
160 C=2
170 ' PROGRAMA PRINCIPAL
180 SCREEN 2:COLOR 7,1,1:CLS
190 VW=0:FOR ANGLE=0 TO 360 STEP SP
200 VW=VW+1 :C=5
210 AG=ANGLE/180*PI
220 RB=R
230 FOR NO=0 TO 5
240 X(NO)=RB*RATIO*COS(AG+A(NO))*RX
250 Y(NO)=RB*RATIO*SIN(AG+A(NO))*RY
260 RB=RB*RATIO
270 NEXT NO
280 FOR NO=0 TO 4
290 LINE(X(NO)+XC,Y(NO)+YC)-(X(NO+1)+XC,
Y(NO+1)+YC),C
300 NEXT NO
310 IF XB<>0 AND FLAG=1 THEN LINE(XB+XC
,YB+YC)-(X(0)+XC,Y(0)+YC),C
320 IF XE<>0 AND FLAG=1 THEN LINE(XE+XC
,YE+YC)-(X(5)+XC,Y(5)+YC),C
330 IF XM<>0 AND FLAG=1 THEN PAINT((XM+
X(4)+XC*2)/2,(YM+Y(4)+YC*2)/2),C
340 XB=X(0):YB=Y(0)
350 XE=X(5):YE=Y(5)
360 XM=X(4):YM=Y(4)
```

```
370 FLAG=-FLAG
380 IF VW=1 THEN 390 ELSE VW=0
390 NEXT ANGLE
400 [illegible] 400:NEXT I:CLEAR:CLS:GOT
O 10
```

PROGRAMA DEMOSTRACION DE GRAFICOS. 2.

```
10 REM **** GRAFICOS ****
20 ' INICIALIZACION
30 DIM R(9),S(9)
40 XS=3
50 XF=2
60 YW=50
70 R(1) 48:R(2) 10:R(3) 5:R(4) 5:R(5) 5:
R(6)-5:R(7) 10:R(8) 48:R(0) 0:S(0) 0
80 FOR NO=1 TO 8
90 S(NO)=S(NO-1)+R(NO)
100 NEXT NO
110 PI=3.14159
120 'PROGRAMA PRINCIPAL
130 SCREEN 2:COLOR 8,1,1:CLS
140 FOR X=36 TO 222 STEP XS
150 Y=YW*SIN(XF*X/180*PI)
160 FOR NO=1 TO 7 STEP 2
170 CIRC [illegible] R(NO *COS(PI/4 ,(S(NO 1)*2+
R(NO )*SIN(PI/4 +Y),R(N( ,,PI*7/4,PI 4
180 CIRC [illegible] +R(NO+1)*COS(PI/4),(S(NO)*2+
R(NO+1))*SIN(PI/4 +Y),R(NO+ ,,PI*3/4,PI
*5/4
190 NEXT NO
200 NEXT X
210 F [illegible] 5000:NEXT I:CLEAR:CLS:GOT
O 20
```

PROGRAMA DE GRAFICOS: 'SPRITES'
MOVIMIENTO DEL CANON - CURSORES
DISPAROS - BARRA DE ESPACIO

```
5 SCREEN 1,0:KEY OFF:COLOR15,4,4:A RND(
TIME)
20 FOR I=0 TO 2
30 GOSUB 3010
40 SPRITE$(I)=SP$
50 NEXT I
70 ON STRIG GOSUB 1010:STRIG(0)ON
80 ON SPRITE GOSUB 2010
90 X0-0:Y0-30:X1=120:Y1=170
100 PUT SPRITE 1,(X1,Y1),15,1
120 ON (STICK(0)+2)/4 GOSUB 200,300
130 X0-X0+INT(RND(1)*5+1)
140 Y0=Y0+INT(RND(1)*11-5)
150 PUT SPRITE 0,(X0,Y0),15,0
160 GOTO 100
200 X1=X1+2:RETURN
300 X1=X1-2:RETURN
1010 SOUND 6,15:SOUND 7,7
1020 SOUND 8,16:SOUND 9,16
1030 SOUND 10,16:SOUND 11,0
1040 SOUND 12,10:SOUND 13,1
1050 STRIG(0) ON:SPRITE ON
1060 X2=X1
1070 FOR Y2=Y1-8 TO -8 STEP -1
1080 PUT SPRITE2,(X2,Y2),15,2
1090 NEXT Y2
1100 RETURN
2010 SOUND 0,65:SOUND 1,15
2020 SOUND 2,97:SOUND 3,15
2030 SOUND 4,162:SOUND 5,15
2040 SOUND 6,15:SOUND 7,21
2050 SOUND 8,31:SOUND 9,31
2060 SOUND 10,31:SOUND 11,228
2070 SOUND 12,37:SOUND 13,1
2080 PUT SPRITE0,(X0,Y0),8,0
2090 LOCATE 10,12:PRINT"GAME OVER"
```

```
2100 FOR I=1 TO 1000:NEXT:RESTORE:SPRITE
 OFF:CLEAR:GOTO 5
3010 SP$='"
3020 FOR J=1 TO 8
3030 READ D$
3040 SP$-SP$+CHR$(VAL('&B"+D$))
3050 NEXT J
3060 RETURN
4010 DATA 00011000
4020 DATA 00111100
4030 DATA 01111110
4040 DATA 11011011
4050 DATA 11011011
4060 DATA 11111111
4070 DATA 00100100
4080 DATA 01000010
4100 DATA 00010000
4110 DATA 00111000
4120 DATA 00111000
4130 DATA 01111100
4140 DATA 11111111
4150 DATA 10101010
4160 DATA 11111110
4170 DATA 11000110
4190 DATA 00010000
4200 DATA 00010000
4210 DATA 00000000
4220 DATA 00000000
4230 DATA 00000000
4240 DATA 00000000
4250 DATA 00000000
4260 DATA 00000000
```

PROGRAMA DE MUSICA: ORGANO MUSICAL

```
10 REM **** ORGANO MUSICAL ****
20 '
30 '-----------------------------------

40 ' TECLAS:.QWERTYU  --> DO-SI(OCTAVA 2)

50 '         .I        --> DO   (OCTAVA 3)

60 '         .ASDFGHJ --> DO-SI(OCTAVA 4)

70 '         .K        --> DO   (OCTAVA 5)

80 '         .ZXCVBNM --> DO-SI(OCTAVA 6)

90 '         .,(COMA) --> DO   (OCTAVA 7)

100 ' -----------------------------------
-
110 '
120 SCREEN 2:CLS:COLOR 8,1,1
130 OPEN"GRP:"FOR OUTPUT AS #1
140 A$="U20R20D20L20"
150 FOR X=20 TO 190 STEP 25
160 PRESET(X,40):DRAW "C5XA$;"
170 NEXT
180 PRESET(20,70):DRAW "C5XA$;"
190 FOR X=20 TO 190 STEP 25
200 PRESET(X,100):DRAW "C5XA$;"
210 NEXT
220 PRESET(20,130):DRAW"C5XA$;"
230 FOR X=20 TO 190 STEP 25
240 PRESET(X,160):DRAW"C5XA$;"
250 NEXT
260 PRESET(20,190):DRAW"C5XA$;"
270 FOR X=27 TO 190 STEP 25
280 READ A$
290 PRESET(X,27):PRINT#1,A$
300 NEXT
310 PRESET(27,57):READ A$:PRINT#1,A$
320 FOR X=27 TO 190 STEP 25
330 READ A$
```

```
340 PRESET(X,87):PRINT#1,A$
350 NEXT
360 PRESET(27,117):READ A$:PRINT#1,A$
370 FOR X=27 TO 190 STEP 25
380 READ A$
390 PRESET(X,147):PRINT#1,A$
400 NEXT
410 PRESET(27,177):READ A$:PRINT#1,A$
420 COLOR 14
430 FOR X=23 TO 185 STEP 25
440 READ A$
450 PRESET(X,8):PRINT#1,A$
460 NEXT
470 PRESET(200,8):PRINT#1,"OCTAVA"
480 COLOR 8
490 FOR Y=27 TO 177 STEP 30
500 PRESET(210,Y):READ A:PRINT#1,A
510 NEXT
520 PRESET(50,57):COLOR 2:PRINT#1,"ORGA
NO MUSICAL"
530 PRESET(70,117):PRINT#1," ♪ ♩ ♩ ♪"
540 GOSUB 900
550 PLAY"L15V15"
560 A$=INKEY$:IF A$="" GOTO 560
570 IF A$="Q" THEN PLAY"O2C"
580 IF A$="W" THEN PLAY"O2D"
590 IF A$="E" THEN PLAY"O2E"
600 IF A$="R" THEN PLAY"O2F"
610 IF A$="T" THEN PLAY"O2G"
620 IF A$="Y" THEN PLAY"O2A"
630 IF A$="U" THEN PLAY"O2B"
640 IF A$="I" THEN PLAY"O3C"
650 IF A$="A" THEN PLAY"O4C"
660 IF A$="S" THEN PLAY"O4D"
670 IF A$="D" THEN PLAY"O4E"
680 IF A$="F" THEN PLAY"O4F"
690 IF A$="G" THEN PLAY"O4G"
700 IF A$="H" THEN PLAY"O4A"
710 IF A$="J" THEN PLAY"O4B"
720 IF A$="K" THEN PLAY"O5C"
730 IF A$="Z" THEN PLAY"O6C"
740 IF A$="X" THEN PLAY"O6D"
750 IF A$="C" THEN PLAY"O6E"
760 IF A$="V" THEN PLAY"O6F"
```

```
770 IF A$="B" THEN PLAY"O6G"
780 IF A$="N" THEN PLAY"O6A"
790 IF A$="M" THEN PLAY"O6B"
800 IF A$="." THEN PLAY"O7C"
810 GOTO 560
820 DATA Q,W,E,R,T,Y,U
830 DATA I
840 DATA A,S,D,F,G,H,J
850 DATA K
860 DATA Z,X,C,V,B,N,M
870 DATA ","
880 DATA DO,RE,MI,FA,SOL,LA,SI
890 DATA 1,2,3,4,5,6
900 REM **** MUSICA INICIAL ****
910 '
920 A$="L8O6C4.ECEG16L64CFAL8O7CO6BAGG5.
.O5L64GBO6DL8GFDO5BGBO6DFECAGG4
930 B$="O5L8CGEG"
940 C$="O5CAFA"
950 D$="O4GO5FDF"
960 F$=B$+B$+C$+B$+D$+D$+B$+B$
970 T$="L8O6C4.ECEG16L64CFAL8O7CO6BAGG5.
.O5L64GBO6DL8GFDO5BGBO6DO5BO6CO5GO6ECC2

980 G$=B$+B$+C$+B$+D$+D$+B$+B$
990 PLAY"S1M3000T120","S8M500T120":PLAY
A$,F$:PLAY T$,G$
1000 RETURN
```

PROGRAMA DE MUSICA: HIMNO

```
10 REM **** PROGRAMA MUSICAL ****
20 REM          H I M N O
30 COLOR 4,14,14:CLS:GOSUB 100:CLEAR:PLA
Y"V12","V12","V12"
40 V1$="T108O6L4CCCO5G.O6C8DEEE.D8EE8L8
CO5GLEE.D8F8F8CGGG.F8FFFF.F8DEL8FF.DCL4E
.F8GL12AGFL4EDCR4R4 R20"
50 V2$="T108O5L4EGAD.G8GGAACBAFFDF.R8R4C
GGG.G8GCCGG.G8GGL8AGFEL4G.F8GR4GFER4R4 R
20"
```

```
60 V3$="T108O4L4DEFG.A8BC[illegible]O4AFG#AFGO3G
O4[illegible]O3CR4.4.FGO5C[illegible]4[illegible]R4[illegible]4.[illegible]3CR4 4CC5
CC.O4A8EF8D8GO3GO4CO3GC R20"
70 PLAY V1$,V2$,V3$
80 FOR F=1 TO 8000:NEXT F
90 GOTO 70
100 SCREEN 0:PRINT:PRINT:PRINT:PRINT:PRI
NT:PRINT:PRINT:PRINT
110 PRINT"              ┌──────────────────┐"
120 PRINT"              │                  │"
130 PRINT"              │    H I M N O     │"
140 PRINT"              │                  │"
150 PRINT"              └──────────────────┘"
160 RETURN
```

PROGRAMA DE MUSICA: 'ROCK'

```
10 REM ****** MUSICA ******
20 REM  EFECTOS ESPECIALES
30 REM      **  ROCK  **
40 CLS:KEY OFF:CLEAR:COLOR 1,INT(RND(1)*
15)+2
50 LOCATE 14,17:PRINT"R O C K"
60 B=500:PLAY"V12"
70 A$="T230O3EG#B.O4C#DC#O3BG#"
80 B$="T230O3AO4C#EF#GF#EC#"
90 C$="T230O3BO4D#F#G#AG#F#D#"
100 FOR A=3 TO 15
110 IF A=9 OR A=5 OR A=6 OR A=7 OR A=15
THEN 150
120 PLAY"M=B;":PLAY"S=A;"
130 FOR I=1 TO 2:PLAY A$:COLOR,INT(RND(1
)*14)+2:NEXT
140 PLAY B$:COLOR,INT(RND(1)*14)+2:PLAY
A$:COLOR,INT(RND(1)*14)+2:PLAY C$:COLOR,
INT(RND(1)*14)+2:PLAY B$:COLOR,INT(RND(1)
*14)+2
150 NEXT A:GOTO 100
```

PROGRAMA PARA HACER UNA COPIA DE LA PANTALLA (SCREEN 0) POR IMPRESORA

```
10 REM *** HARD-COPY ->SCREEN 0 ***
20 '
30 FOR K=0 TO 960
40 A=VPEEK(BASE(0)+K)
50 LPRINT CHR$(A);
60 IF L=39 THEN 80 ELSE IF L<39 THEN L=L
+1
70 NEXTK:END
80 LPRINT:L=0:GOTO 70
```

PROGRAMA PARA HACER UNA COPIA DE LA PANTALLA (SCREEN 1) POR IMPRESORA

```
10 REM *** HARD-COPY ->SCREEN 1 ***
20 '
30 FOR K=0 TO 767
40 A=VPEEK(BASE(5)+K)
50 LPRINT CHR$(A);
60 IF L=31 THEN 80 ELSE IF L<31 THEN L=L
+1
70 NEXTK:END
80 LPRINT:L=0:GOTO 70
```

PROGRAMA PARA HACER UNA COPIA DE LA PANTALLA (SCREEN 2/3) CON EL PLOTTER PRN-C41 DE SONY.

```
10 '   ***** HARD-COPY *****
20 '
30 '   PLOTTER PRN-C41 SONY
40 '
50 '   SCREEN 2/SCREEN 3
60 '
70 '
```

```
80 COLOR 4,4
90 LPRINT
100 LPRINT CHR$(&H1B)+"#"
110 FOR Y-0 TO 192
120 FOR X=0 TO 250
130 IF POINT(X,Y)=4 THEN LPRINT"R3,0":GO
TO 160
140 LPRINT'J0,3,3,0,0,-3,-3,0
150 LPRINT'R3,0'
160 NEXT X
170 LPRINT'R-753,-3"
180 NEXT Y
```

PROGRAMA PARA UTILIZAR CON EL PLOTTER PRN-C41 DE SONY: 'CIRCULO'

```
10 ' *** PLOTTER PRN-C41 ***
20 '            SONY
30 '       C I R C U L O
40 LPRINT CHR$(27);"#'
50 LPRINT'R125,-125'
60 LPRINT'I'
70 LPRINT'M80,0"
80 P 3.14159
90 A=1:B=1
100 FOR R=0 TO 360 STEP 5
110 S=R/180*P
120 SI=INT(SIN(A*S)*80)
130 CO=INT(COS(B*S)*80)
140 LPRINT"D";CO;",";SI
150 NEXT
160 LPRINT'R0,-125
170 LPRINT"A"
```

PROGRAMA PARA UTILIZAR CON EL PLOTTER PRN-C41 DE SONY: 'CUADRICULA'

```
10 ' **** PLOTTER PRN-C41 ****
20 '              SONY
30 '    CUADRICULA DE COLORES
40 LPRINT
50 LPRINT CHR$(27);"#"
60 LPRINT"I":C=0
70 FOR I=1 TO 20
80 LPRINT"C";
90 IF I MOD 5=0 THEN C=C+1
100 IF C>3 THEN C÷0
110 LPRINT"J198.0"
120 LPRINT"R0.-5"
130 LPRINT [illegible]198.0
140 LPRINT"R0.-5"
150 NEXT
160 LPRINT"R0.5"
170 FOR I=1 TO 20
180 LPRINT"C";C
190 IF I MOD 5=0 THEN C-C+1
200 IF C>3 THEN C-0
210 LPRINT"J0,198"
220 LPRINT"R5,0"
230 LPRINT"J0,-198"
240 LPRINT"R5,0"
250 NEXT
260 LPRINT:LPRINT
270 LPRINT"A"
```

JUEGO: LABERINTO

```
5 L-0
6 C.EAR:RESTORE:COLOR 7.1.1
10 'LABERINTO'
11 CLS:SCREEN 1:PRINT:PRINT:PRINT:PRINT
12 PRINT
13 PRINT"              LABERINTO  "
14 PRINT"                          "
```

```
15 FOR I=1 TO 1000
16 NEXT I
20 ON ERROR GOTO 470
30 DEFINT A-Z:DEF FNE(X,Y)=((X=WX-1) AND
  Y=WY-1))
40 DIM DX(3),DY(3):FOR I=0 TO 3:READ DX(
I),DY(I):NEXT
50 DATA 2,0,0,2,-2,0,0,-2
60 LOCATE 7,8: PRINT"NIVEL ? (1-3)":A$=I
NKEY$:L=VAL(A$):IF L<1 OR L>3 GOTO 60
70 FOR I=1 TO L:READ WX:NEXT :WY=WX
80 DATA 5,8,12
90 FOR I=1 TO VAL(RIGHT$(STR$(TIME),2)):
X=RND(1):NEXT
91 PRINT:PRINT:PRINT:PRINT
100 PRINT"ESTAS EN UN LABERINTO, DEBES ":
PRINT:PRINT"     BUSCAR LA SALIDA":PRINT:
PRINT:PRINT:PRINT"   UTILIZA LOS CURSORES
 PARA":PRINT:PRINT"  MOVERTE POR LOS PAS
ILLOS":KX=WX+1:KY=WY+1:WX=WX*2:WY=WY*2
110 DIM M(WX,WY)
120 FOR I=0 TO WX:M(0,I)=1 :M(I,0)=1 :M(
WX,I)=1:M(I,WY)=1:NEXT
130 C=(KX-2)*(KY-2):CA=C-4
140 X=INT(RND(1)*KX)*2:Y=INT(RND(1)*KY)*
2 :IF M(X,Y)=0 GOTO 140
150 GOSUB 260:IF C<CA THEN  IF CN GOTO 1
50
160 IF C<>0 GOTO 140
170 'JUEGO
180 SCREEN 2:XM=1:YM=1:DR=0
190 GOSUB 320 'LINE
200 IF FNE(XM,YM) THEN  OPEN "GRP:"FOR O
UTPUT AS #1:PSET(90,10):PRINT#1,"S A L I
 D A":GOSUB 500:FOR QW=1 TO 1500:NEXT QW
:SCREEN 1:LOCATE 1,10: PRINT "  HAS ENCO
NTRADO LA SALIDA":GOSUB 600
210 A=STICK(0):IF A=0 THEN 210
220 A=(A-1)/2:DR=(DR + A) MOD 4
230 IF M(XM+DX(DR)/2,YM+DY(DR)/2) THEN D
R=(DR+4-A) MOD 4:GOTO 210
240 XM=XM+DX(DR):YM=YM+DY(DR)
250 GOTO 190
260 ' [illegible]
```

```
270 R0=INT(RND(1)*4):R=R0
280 IF M(X+DX(R),Y+DY(R))=0 GOTO 300
290 R=(R+1)MOD 4:IF R<>R0 GOTO 280 ELSE
CN=0:RETJRN
300 M(X+DX(R)/2,Y+DY(R)/2)=1:X=X+DX(R):Y
=Y+DY(R)
310 M(X,Y)=1:C=C-1:CN=-1:RETURN
320 CLS
330 X=XM:Y=YM:X0=128:Y0=92:LX=128:LY=92
340 J0=DX(DR)/2:V0=DY(DR)/2
350 J1=DX((DR+1) MOD 4)/2:V1=DY((DR+1) M
OD 4)/2:J2=DX((DR+3) MOD 4)/2:V2=DY((DR+
3) MOD 4)/2
360 IF M(X,Y) THEN IF FNE(X+J0,Y+V0) GO
TO 450 ELSE 460
370 IF M(X+J1,Y+V1) THEN GOSJB 430 ELSE
GOSJB 400
380 LX=LX:IF M(X+J2,Y+V2) THEN GOSJB 43
0 ELSE GOSUB 400
390 LX=ABS(LX*.8):LY=LY*.8:X=X+J0:Y=Y+V0
:GOTO 360
400 LINE(X0+LX,Y0+LY)-(X0+LX,Y0+LY)
410 LINE(X0+LX,Y0+LY*.8)-(X0+LX*.8,Y0+LY
*.8)
420 LINE (X0+LX*.8,Y0-LY*.8):LINE -(X0+L
X,Y0-LY*.8):RETURN
430 LINE(X0+LX,Y0+LY)-(X0+LX*.8,Y0+LY*.8
)
440 LINE(X0+LX,Y0-LY)-(X0+LX*.8,Y0-LY*.8
):RETLRN
450 LINE(X0-LX,Y0-LY)-(X0+LX,Y0+LY),,BF:
RETURN
460 LINE(X0-LX,Y0-LY)-(X0+LX,Y0+LY),,B:R
ETJRN
470 IF ERL=280 THEN IF ERR=5 OR ERR=9 TH
EN RESUME 290
480 ON ERROR GOTO 0
490 END
500 FOR I=1 TO 3:PLAY"T200O5L62GECGECGEC
GECGECGEC":NEXT I:RETURN
600 LOCATE 2,15:PRINT"QUIERES CONTINUAR
S/N)";:A$=INKEY$:IF A$="S" THEN 5 ELSE I
F A$="N" THEN END ELSE IF A$<>"S" OR A$<>
"N" GOTO 600
```

PROGRAMA PARA DIBUJAR CON LOS CURSORES

```
10   ON STOP GOSUB 250:STOP ON
20 CLS
30 COLOR 1,15,15
40 SCREEN 2,0
50 OPEN 'GRP:'FOR OUTPUT AS #1
60 PRESET(17,1)
70 PRINT #1,"TECLAS DE COMANDO PARA DIBU
JAR
80 PRESET(17,32)
90 PRINT#1,"F1:PALETA DE COLORES(selecci
on  con cursor)"
100 PRESET(17,56)
110 PRINT#1,"F2:LINEAS (longitud y posic
 on  final con cursor)
120 PRESET(17,80)
130 PRINT#1,"F3: CIRCUNFERENCIAS(forma y
 ta  maño con cursor)"
140 PRESET(17,104)
150 PRINT#1,"F4:CUADRADOS COLOREADOS (fo
rma  y tamaño con cursor)"
160 PRESET(17,128)
170 PRINT#1,"F5:COLOR A SUPERFICIES(solo
 si  estan cerradas)
180 PRESET(17,152)
190 PRINT#1,"F6:BORRADO DEL DIBUJO"
200 PRESET(17,170)
210 PRINT#1,"PARA EJECUTAR COMANDOS PULS
E'
220 PRESET(17,183)
230 PRINT#1,"TECLA RETURN'
240 A$=INKEY$:IF A$=""THEN240
250 CLS
260 COLOR 1,15,15
270 SCREEN2
280 FOR D=0TO249STEP16
290 LINE(D,187)-(D+16,190),D/16,BF
300 NEXTD
310 SPRITE$(1)=CHR$(&H0)+CHR$(&H10)+CHR$
(&H10)+CHR$(&H7C)+CHR$(&H10)+CHR$(&H10)+
CHR$(&H0)+CHR$(&H0)
320 C 0
```

```
330 X=128:Y=96
340 PUT SPRITE 3,(X-3,Y-4),1,1
350 A$=INKEY$
360 PSET(X,Y),C
370 ON KEY GOSUB 650,500,780,940,620,109
0
380 KEY(1)ON:KEY(2)ON:KEY(3)ON:KEY(4)ON:
KEY(5)ON:KEY(6)ON
390 IF A$="" THEN 340
400 IF X=256 THEN X=X-1
410 IF Y=186 THEN Y=Y-1
420 IF X=0 THEN X=X+1
430 IF Y=0 THEN Y=Y+1
440 IF A$=CHR$(28) THEN X=X+1
450 IF A$=CHR$(29) THEN X=X-1
460 IF A$-CHR$(30) THEN Y=Y-1
470 IF A$=CHR$(31) THEN Y=Y+1
480 IF A$=CHR$(84) THEN 600
490 GOTO 340
500 A=X:B-Y
510 PSET(X,Y),C
520 A$ INKEY$
530 IF A$-CHR$(13)THEN RETURN
540 IF A$=""THEN520
550 LINE(A,B)-(X,Y),15
560 IF A$=CHR$(28)THENX=X+1
570 IF A$ CHR$(29)THENX=X-1
580 IF A$ CHR$(30)THENY=Y-1
590 IF A$=CHR$(31)THENY=Y+1
600 LINE(A,B)-(X,Y),C
610 GOTO 520
620 X=X:Y=Y
630 PAINT(X+2,Y+2),C
640 RETURN
650 N=0
660 LINE(N,186)-(N+16,192),1,B
670 C=POINT(N+5,190)
680 A$=INKEY$
690 IF A$="" THEN 680
700 LINE(N,186)-(N+16,192),15,B
710 IF A$=CHR$(13)THEN RETURN
720 IF A$=CHR$(28)THENN-N+16
730 IF A$=CHR$(29)THENN=N-16
```

```
740 IF N=-16THENN-0
750 IF N=256THENN=N-16
760 LINE(N,186)-(N+16,192),1,B
770 GOTO660
780 F=X:G=Y
790 A$=INKEY$
800 [illegible]:B=ABS(G-Y[illegible]A+1:[illegible]B+1:
ABS([illegible]1)
810 [illegible]
820 IF A$=""THEN790
830 IF A$=CHR$(13)THEN RETURN
840 CIRCLE(X,Y),A,15,,,T
850 IF A$=CHR$(28)THENF=F+1
860 IF A$-CHR$(29)THENF=F-1
870 IF A$-CHR$(30)THENG-G-1
880 IF A$=CHR$(31)THENG=G+1
890 IF F-0 THENF=F+1
900 IF F=255 THENF-F-1
910 IF G-0 THENG-G+1
920 IF G=185 THENG-G-1
930 GOTO 790
940 D=X:E=Y
950 A$=INKEY$
960 IF A$=CHR$(13)THEN RETURN
970 IF A$=""THEN950
980 LINE(D,E)-(X,Y),15,BF
990 IF A$=CHR$(28)THENX=X+1
1000 IF A$=CHR$(29)THENX=X-1
1010 IF A$=CHR$(30)THENY=Y-1
1020 IF A$-CHR$(31)THENY=Y+1
1030 IF X=0 THENX=X+1
1040 IF X-255 THENX=X-1
1050 IF Y=0 THENY=Y+1
1060 IF Y=185 THENY=Y-1
1070 LINE(D,E)-(X,Y),C,BF
1080 GOTO 950
1090 GOTO 250
```

ANEXO 5: MENSAJES Y CODIGOS DE ERROR

Código	Mensaje	Explicación
1	NEXT Without FOR	Existe un nombre de variable en una instrucción NEXT que no corresponde a la variable de la ins trucción FOR o no existe el NEXT.
2	Syntax error	Una linea de programa contiene caracteres incorrectos (puntuación, parentesis no emparejados, error mecanográfico ..)
3	RETURN without GOSUB	Se encontro una instrucción RETURN sin que se hubiera incorporado un comando GOSUB GOSUB no emparejado
4	Out of DATA	Se efectuó una instruccion READ, pero no quedaban datos por leer en la instruccion DATA
5	Illegal function call	Un parametro fuera de gama se ha asignado a una funcion matematica o de STRING Un Illegal function call puede ocurrir tambien como resultado de 1 Una instruccion errónea o excesivamente larga 2. Un argumento negativo o cero con LOG 3. Un argumento negativo con SQR 4 Un argumento impropio para MID $ LEFT $ RIGHT $ INP OUT PEEK, POKE TAB, SPC, STRIN $, SPACE $, INSTR $ o ON GOTO

6	Overflow	El resultado de un calculo es demasiado largo para ser representado en el formato BASIC
7	Out of memory	Un programa es demasiado largo tiene demasiadas instrucciones demasiados GOSUB demasiados FOR demasiadas variables o expresiones demasiado complejas
8	Undefined line number	Una linea referenciada en un GOTO GOSUB IF... THEN... ELSE es una linea inexistente
9	Subscript out of range	Un elemento de la matriz esta referenciado con una cifra fuera de las dimensiones de la matriz o posee un numero erróneo de subíndices
10	Redimensioned array	Una matriz puede dimensionarse solo una vez, con DIM Si se efectuan dos DIM para una misma matriz aparece este mensaje si se usa una variable de matriz antes de que esta sea dimensionada, se ejecuta una operacion DIM automaticamente dimensionandose en 10 elementos Si se pretende dimensionar posteriormente se generará el mensaje de error citado
11	Division by zero	Al ser una operación sin sentido matematico aparece el mensaje de error
12	Illegal direct	Una instruccion que solamente puede incorporarse en un programa se ha introducido en Modo directo
13	Type mismatch	Un numero se ha asignado a una cadena o vice versa una funcion que esperaba un numero recibe una cadena o viceversa
14	Out of string space	Unas variables string han sobrepasado la memoria disponible El BASIC asignara dinamicamente espacio para cadenas hasta que se agoten los recursos de memoria
15	String too long	Aparece este mensaje si se pretende abarcar en una cadena mas de 255 caracteres
16	String formula too complex	La expresion de la cadena es demasiado larga o compleja Deberia div dirse al menos en dos partes para que pueda ser elaborada.

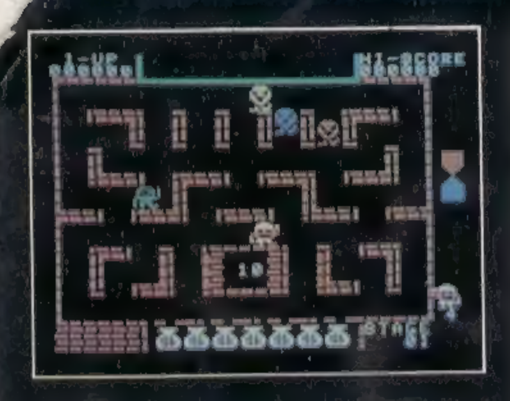
17	Can t continue.	El programa no puede continuar porque 1. Se ha producido un error. 2 se ha modificado durante una interrupcion de su ejecucion 3 No existe
18	Undefined user function	Se uso como referencia una funcion definida por el usuario pero la misma no hab a sido definida usando la instrucción DEF FN
19	Device I/0 error	Error al conectar cassette, impresora, pantalla Es un error *fatal* ya que el programa no puede recuperarse
20	Verify error	El programa actual es distinto del grabado en cassette. (Grabación incorrecta)
21	NO RESUME	Se ha entrado una rutina de tratamiento de errores, pero no hay ninguna instrucción RESUME
22	RESUME without error	Se encuentra una instruccion RESUME antes de que se haya entrado en una rutina de tratamiento de errores
23	Unprintable error	No hay mensaje de error por la condición de error que se da Usualmente es el caso de una instruction ERROR con un codigo de error no definido.
24	Missing operand	Una expresion contiene un operador no seguido de operando
25	Line buffer overflow	Se ha entrado una linea con demasiados caracteres.
26-49		Estos códigos no tienen definición Se reservan para futuras expansiones del BASIC
50	Field overflow	Una instruccion FIELD intento colocar más bytes de los especificados para la longitud de registro de un fichero aleatorio
51	Internal error	Se ha producido un funcionamiento defectuoso en el lenguaje BASIC
52	Bad file number	En una instrucción o comando se cita un numero de fichero que no se ha abierto anteriormente o que esta fuera de la gama de numero de fichero especificados en la instrucción MAXFILES

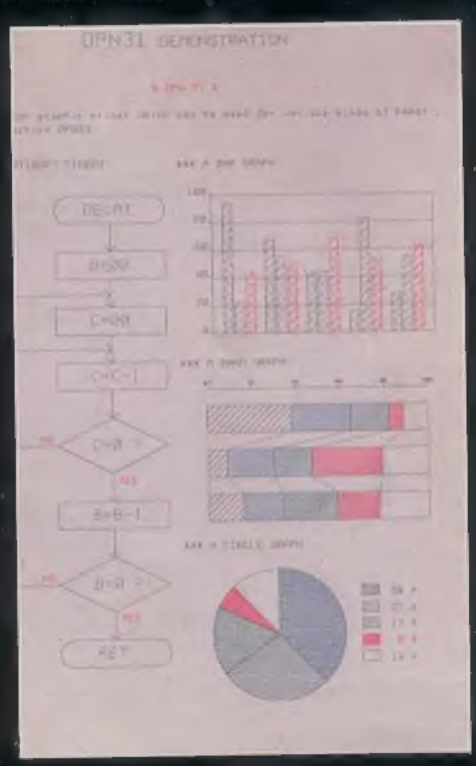
53	File not found	En una instruccion LOAD, OPEN o KILL se cita un fichero que no existe en la memoria
54	File already open	Se ha encontrado una instrucción de salida para un fichero que ya estaba abierto, o se ha entrado un comando KILL para un fichero que está abierto
55	Input past end	Se ha ejecutado una instrucción INPUT tras haberse entrado todos los datos en el fichero o a partir de un fichero nulo Para evitar este error usar la función EOF para detectar el final del fichero.
56	Bad file name	Se ha usado una forma ilegal de nombre de fichero p.e.: LOAD, SAVE, KILL, NAME, etc.
57	Direct statement in file	Se encuentra una instruccion directa al cargar con LOAD un fichero en formato ASCII La ejecución de LOAD se ha interrumpido
58	Secuential I/O only	Una instruccion de acceso aleatorio es utilizada para un fichero secuencial
59	File not open...	El fichero designado con PRINT #, INPUT #, etc no se ha abierto
60-255		Estos codigos no tienen definicion El usuario puede crear su propio codigo de errores dentro de esta secuencia








1-UP 1
HI-SCORE






STAGE-01


Expenses for month of May
(Yen base)
Total budget= yen 250,000
date food elec educ

MSX

SONY®